Parallel Optimization of NTT for Kyber Key Encapsulation Mechanism Using GPU-Accelerated Plantard Arithmetic

Muhammad Asfand Hafeez, Wai-Kong Lee, *Member, IEEE*, Arslan Munir, *Senior Member, IEEE*, and Seong Oun Hwang, *Senior Member, IEEE*

***Abstract*—CRYSTALS-Kyber is a key encapsulation mechanism that has been standardized by the National Institute of Standards and Technology. Number theoretic transform (NTT) is used to compute the polynomial multiplication in Kyber, which is also the main performance bottleneck. In this letter, a GPU-optimized implementation of Plantard arithmetic intended for improving the performance of NTT is introduced. Various techniques are proposed to avoid warp divergence and expensive division in Plantard arithmetic, thus achieving a faster implementation. Experimental outcomes reveal performance improvements of 6.57%, 6.88%, and 6.76% for NTT, as well as 2.18%, 2.67%, and 3.24% for inverse NTT (INTT), when compared to Lee et al., for Kyber-512, Kyber-768, and Kyber-1024, respectively. These results underscore the efficient and scalable implementation of Kyber, rendering it particularly suitable for the Internet of things platforms where high throughput and moderate latency is imperative.**

***Index Terms*—Post-Quantum cryptography, Internet of things, Plantard multiplication, Number Theoretic Transform**

I. Introduction

DUE to the rapid development of quantum computing, traditional cryptographic schemes that were once considered secure are now facing significant threats. In 2023, the National Institute of Standards and Technology (NIST) completed its standardization process and selected Kyber [1] as a standard for key encapsulation mechanism (KEM). However, like many other post-quantum cryptography (PQC) algorithms, Kyber also faces a significant challenge in practical implementation, especially the polynomial multiplication in a finite ring. Addressing this performance issue is essential for the widespread adoption of Kyber in various applications, such as secure communications and data protection in the Internet of things (IoT). Due to this reason, substantial efforts have been directed toward enhancing its versatility through efficient implementations across various platforms, including field-programmable gate arrays (FPGAs) [2], [3], microcontrollers [4], [5], and graphics processing units (GPUs) [6], [7].

Gupta et al. [7] were among the pioneers in presenting a parallel implementation of Kyber on a GPU that achieved high throughput. Subsequently, Lee et al. [8] successfully implemented both Kyber and New Hope KEM on the GPU platform, using a fully parallel NTT implementation combined with a level-combination approach, thereby delivering a low latency performance. More recently, Ji et al. [6] introduced a high-performance implementation of Kyber, utilizes three methods: sliced layer merging (SLM), sliced depth-first search (SDFS-NTT), and entire depth-first search (EDFS-NTT), each of which demonstrates significant advancements over previous implementations. However, all these efforts are based on the NTT using Montgomery multiplication and barret reduction.

In this letter, techniques are proposed to improve modular multiplication in Kyber's NTT using Plantard arithmetic [9]. The contributions are summarized below:

1) Building upon the work in [10], we introduce an optimized GPU-based implementation of Plantard arithmetic targeting the computation of NTT in the Kyber KEM. A carefully designed implementation was introduced to avoid warp divergence, thus achieving a better performance compared to previous work.
2) We evaluate the performance of the proposed method using RTX 4080 GPU by benchmarking it against Montgomery multiplication across various Kyber variants (Kyber-512, Kyber-768, and Kyber-1024). The proposed method achieves speedup of 6.57%, 6.88%, and 6.76% for NTT and 2.18%, 2.67%, and 3.24% for INTT, respectively.
3) The code is available at github for public evaluation.

II. Proposed Method

A. Plantard Arithmetic

Plantard arithmetic is designed to operate with word-sized moduli, utilizing precomputed constants to reduce the necessity of expensive division operations. For a given modulus Q, this method precomputes $Q^{-1} \mod 2^{2l}$ (where l refers to the word size, such as 16 or 32 bits) and employs this value to manage the reduction of intermediate results efficiently. Recently, Huang et al. [10] proposed an improved version of Plantard arithmetic that not only accommodates larger input ranges but also narrows the output range. This enhancement facilitates efficient modular operations while minimizing memory and computational overhead. By adapting the arithmetic to effectively handle signed integers, the need for additional additions to manage negative outputs has been eliminated, thereby streamlining computational processes. The input range for multiplication has been extended from $[0, Q]$ to $[-Q2^{\alpha}, Q2^{\alpha}]$, with $\alpha > 0$ ensuring a more extensive input space. This expansion decreases the frequency of modular reductions and allows for enhanced "lazy reduction" strategies. Furthermore, in a subsequent study by Huang et al. [11], the authors present an optimized version of the algorithm

Muhammad Asfand Hafeez and Arslan Munir are with the Department of Electrical Engineering and Computer Science, Florida Atlantic University, Boca Raton 33431, USA. (e-mail: mhafeez2024@fau.edu, arslanm@fau.edu)

Wai-Kong Lee is with the Faculty of Information and Communication Technology, Universiti Tunku Abdul Rahman, Malaysia (e-mail: wklee@utar.edu.my).

Seong Oun Hwang is with the Department of Computer Engineering, Gachon University, Seongnam 13120, South Korea (sohwang@gachon.ac.kr)

Algorithm 1 Optimized `Plantard_Multiplication`

Require: a, b: 32-bit signed integers
Ensure: t: Intermediate result after multiplication and scaling
// `Compute high 16 bits of` $a \times b$ *Using GPU Intrinsics*
1: $t \leftarrow$ `__mulhi`(a, b)
2: $t \leftarrow t + 8$ *Adjust for rounding*
3: $t \leftarrow t \times Q$
4: $t \leftarrow t \gg 16$
5: **return** t

Algorithm 2 Optimized `Plantard_Reduction`

Require: a: 32-bit signed integer
Ensure: t: Result after modular reduction, $0 \leq t < Q$
1: $t \leftarrow$ `__mulhi`(a, Q_inv_Plant) *Compute approximate quotient*
2: $t \leftarrow t \times Q$
3: $t \leftarrow a - t$
4: if $(t < 0)$ $t+ = Q)$ (see Table I)
5: **return** t

that maximizes the use of available instructions on low-end platforms, such as ARM Cortex-M3 and RISC-V, without reliance on SIMD extensions or specialized hardware.

B. Proposed Optimized Plantard Arithmetic

The proposed optimized Plantard arithmetic builds upon the work from [10], specifically optimized for parallel implementation of NTT operations on GPUs. The optimized approach executes multiplication as detailed in Algorithm 1, utilizing GPU intrinsics to calculate the scaled intermediate result with minimal overhead. High-performance operations are employed to keep intermediate values within a manageable range, thus minimizing redundant computations while ensuring precision.

Algorithm 1 illustrates signed 32-bit multiplication of two integers a and b. In previous implementations, the high-order bits of the product $a \cdot b'$ were extracted through explicit division by 2^{16}, where b' is precomputed as $b \cdot Q_{\text{inv}} \mod 2^{2l}$. In our work, we instead utilize the GPU intrinsic function `__mulhi`, which efficiently computes the high-order bits from the signed 32-bit multiplication of a and b, thus avoiding costly division and improving performance. This intrinsic operation is particularly advantageous on GPU platforms where minimizing instruction latency is critical. Subsequently, a rounding constant is added before scaling by the modulus Q (lines 2 and 3) [10]. Since right-shifting by 16 bits discards lower bits, potentially introducing rounding errors, a constant value of $+8$ is added to t to compensate. This value enables unbiased rounding by accounting for the discarded fractional portion, thus preventing systematic bias and aligning the intermediate result. The final scaling step applies an additional right shift, accurately calibrating the intermediate value for multiplication (lines 4 and 5) and yielding the final reduced product.

C. Proposed Techniques to Avoid Warp Divergence on GPUs

This section presents the optimization techniques specifically designed for GPU architectures to enhance the efficiency of Plantard arithmetic, which is different from prior work [10].

TABLE I: Comparison of conditional vs. proposed branchless implementation

Conditional Branching	Proposed approach
`if (t < 0) t += Q`	`t = t + ((t >> 31) & Q)`
Causes warp divergence	Threads execute uniformly

Algorithm 2 shows the proposed Plantard reduction implementation on GPUs. It performs a reduction by utilizing a precomputed modular inverse, Q_{inv}, to approximate division through multiplication and bit-shifting, thus avoiding the costly division operations. The process begins by efficiently reducing a 32-bit integer modulo Q using the precomputed modular inverse Q_{inv} (line 1). By doing this, we eliminate the need for explicit division. Following this the reduction process is carried out by multiplying the input value a by Q_{inv} and extracting the higher 32 bits of the product using the GPU intrinsic instruction (line 1). This operation approximates the quotient of $\frac{a}{Q}$. Next, it multiplies the quotient by Q and subtracts this product from the original input a, effectively performing a modular reduction (lines 2-3). To ensure that the result is within the range $[0, Q-1]$, the algorithm adjusts the value by adding Q if the result is negative (line 4). This ensures correctness while avoiding branching, maintaining both efficiency and GPU compatibility.

Algorithm 2 is designed to ensure that the intermediate result t remains within the bounds of the range $[0, Q-1]$. In the original work [10], this was achieved by approximating the quotient $\left\lfloor \frac{t}{Q} \right\rfloor$ through the high-order bits of the product $t \cdot Q_{\text{inv}}$, followed by the subtraction of $q \cdot Q$ from t. In this letter, the proposed implementation retains this computation but enhances the process by employing the GPU intrinsic `__mulhi` to derive the high bits of $t \cdot Q_{\text{inv}}$, effectively utilizing a hardware-accelerated alternative instead of using a division.

To address the potential for negative values, the original method employs conditional branching (Algorithm 2, line 4), adjusting t by adding Q when $t < 0$. Note that prior implementations utilizing Montgomery multiplication do not have this problem, because the range is sufficiently large to accommodate the lazy reduction without overflow. However, this approach introduces warp divergence in GPU execution, as some threads take the branch while others do not. This forces serialization and reduces overall throughput. This is because GPUs execute instructions in warps of 32 threads, meaning that all threads must execute the same instruction simultaneously. A conditional branch like:

$$if(t < 0) \ \texttt{then} \ \ t+ = Q;$$

It causes some threads to follow the *true* condition while others follow the *false* condition. This splits execution, requiring the GPU to handle one group of threads first and the other later, significantly impacting performance. To address this issue, the proposed approach replaces conditional branching with a branchless correction using bitwise operations:

$$t = t + ((t \gg 31) \& Q).$$

In this context, $t \gg 31$ extracts the sign bit of t, and the application of bitwise AND ensures that Q is only added when

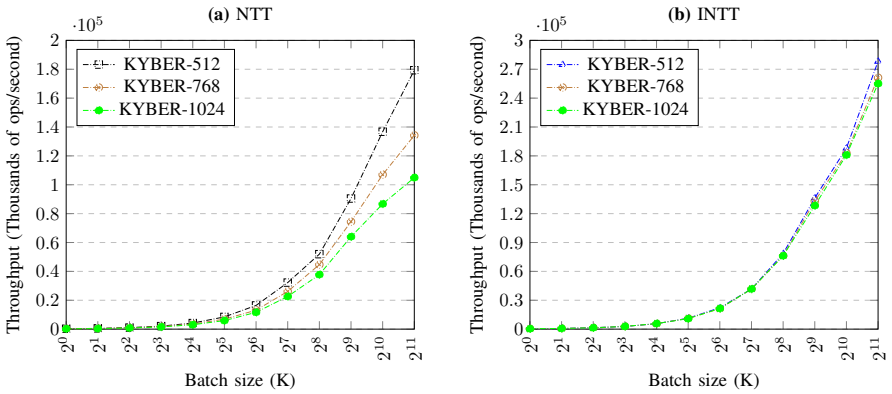

Fig. 1: Throughput analysis of NTT and INTT implementations using Plantard multiplication and reduction for Kyber.

t is negative. This approach greatly reduces warp divergence, a crucial factor for optimizing GPU parallelism. The comparative analysis of both approaches is given in Table I.

1) Validation: The process of combined modular multiplication and reduction can be articulated as follows:

$$t = \left(a \cdot b' \mod 2^{32}\right) \cdot Q \mod Q.$$

This representation illustrates that the intermediate results are maintained within a manageable scale throughout the multiplication, and the reduction step ensures that t is constrained within the range $[0, Q-1]$.

The proposed implementation not only preserves the mathematical correctness of the original methodology but also introduces optimizations tailored for GPU execution. Among these optimizations are the utilization of intrinsics such as `__mulhi` and a branchless reduction technique. These enhancements significantly decrease computational overhead and increase throughput, rendering the implementation highly effective for environments that demand parallel processing capabilities.

III. Experimental Results

A. Throughput analysis of NTT using Plantard Multiplication

This section presents the experimental results of our methodology. The experiments were conducted on a machine equipped with a Core i9 processor operating at 5.8 GHz, along with an Nvidia RTX 4080 GPU. 2048 blocks are instantiated in our implementation, in which each block computes one NTT using $KYBER\_N/2$ threads. In subsequent discussions, we refer the number of blocks as *batch size*.

Figure 1 presents the throughput performance for both NTT and inverse INTT for KYBER-512, KYBER-768 and KYBER-1024, with varying batch sizes. Referring to Figure 1(a), at a batch size of $K = 1$, KYBER-512 exhibits a throughput of 265K operations per second, while KYBER-768 and KYBER-1024 achieve throughputs of 223K and 191K, respectively. As the batch size increases, the throughput increased significantly for all three Kyber variants, demonstrating nearly linear growth in performance. However, at a batch size of $2^9 < K < 2^{10}$, throughput begins to saturate, culminating in a maximum throughput of 179,334K, 134,207K and 104,918K operations per second for KYBER-512, KYBER-768 and KYBER-1024, respectively. Figure 1(b) shows the throughput performance of the INTT implementation for three Kyber variants. The results maintain a similar trend to the NTT analysis, with throughput consistently improving alongside larger batch sizes. The throughput of INTT peaks at 278,227K, 261,224K, and 255,043K operations per second for KYBER-512, KYBER-768, and KYBER-1024 at $K = 2048$. Note that Kyber uses NTT and INTT several times during key generation, encapsulation, and decapsulation. Hence, our results can improve the overall performance of Kyber KEM.

B. Comparison with State-of-the Art NTT Implementations

Table II presents a comprehensive performance comparison between the NTT and its inverse (INTT) utilizing Plantard multiplication and Montgomery multiplication implemented by Lee et al. [8] across several configurations of the Kyber PQC scheme, specifically Kyber-512, Kyber-768, and Kyber-1024. For a fair assessment, the results of [8] were regenerated on RTX 4080. The analysis covers a range of batch sizes (K) from 64 to 2048, highlighting the percentage speedup achieved through the proposed method.

The results demonstrate that Plantard multiplication consistently outperforms Montgomery multiplication across various configurations and batch sizes, with applicability to both the NTT and INTT. For instance, in the case of Kyber-512 at K=64, the NTT implementation using Plantard multiplication takes 3.90 us, compared to 4.10 microseconds for NTT using Montgomery multiplication. This represents an improvement in speed of 5.13% for the Plantard method, which increases to 6.57% at K=2048. This trend is similarly observed for Kyber-768 and Kyber-1024, where the NTT with Plantard multiplication results in speedups of 6.88% and 6.76% at K=2048, respectively.

The INTT exhibits similar results comparable to those of the NTT, although the performance differences are somewhat less pronounced. Plantard multiplication persistently demonstrates faster computation times relative to Montgomery multiplication. For example, at a batch size of K=64 for Kyber-512, the INTT using Plantard multiplication achieves a speedup of 1.04% over Montgomery multiplication, which increases to 2.18% at K=2048. This trend holds true across all variants of Kyber, yielding speedups of 2.67% and 3.24% for Kyber-768 and Kyber-1024, respectively. These findings underscore the computational efficiency of Plantard multiplication in

TABLE II: Performance comparison of the NTT and INTT using Plantard multiplication and reduction with Montgomery

Batch (K)	Kyber-512						Kyber-768						Kyber-1024					
	NTT			INTT			NTT			INTT			NTT			INTT		
	Time (us)																	
	P[1]	M[2]	S[3]	P[1]	M[2]	S[3]	P[1]	M[2]	S[3]	P[1]	M[2]	S[3]	P[1]	M[2]	S[3]	P[1]	M[2]	S[3]
64	3.90	4.10	**5.13**	2.88	2.91	**1.04**	4.88	5.12	**4.92**	3.01	3.05	**1.31**	5.50	5.70	**3.64**	2.98	3.02	**1.34**
128	3.97	4.18	**5.29**	3.07	3.09	**0.65**	4.90	5.18	**5.71**	3.07	3.11	**1.30**	5.66	5.95	**5.12**	3.07	3.14	**2.28**
256	4.93	5.22	**5.88**	3.26	3.30	**1.23**	5.73	6.02	**5.06**	3.36	3.41	**1.49**	6.78	7.14	**5.23**	3.36	3.46	**2.98**
512	5.66	5.98	**5.65**	3.90	3.98	**2.05**	6.88	7.25	**5.38**	3.97	4.03	**1.51**	8.0	8.45	**5.25**	3.84	3.96	**3.12**
1024	7.49	7.96	**6.27**	5.15	5.25	**1.94**	9.57	10.11	**5.64**	5.66	5.75	**1.59**	11.81	12.48	**5.67**	5.63	5.79	**2.84**
2048	11.42	12.17	**6.57**	7.81	7.98	**2.18**	15.26	16.31	**6.88**	7.84	8.05	**2.67**	19.52	20.84	**6.76**	8.03	8.29	**3.24**

[1] Plantard Multiplication; [2] Montgomery Multiplication [8]; [3] Speed up (%)

TABLE III: Execution Time (µs) comparison of Plantard NTT and INTT: Proposed (Branchless) vs With Branch

Batch (K)	Kyber-512				Kyber-768				Kyber-1024			
	NTT		INTT		NTT		INTT		NTT		INTT	
	Proposed	Branch	Proposed	Branch	Proposed	Branch	Proposed	Branch	Proposed	Branch	Proposed	Branch
512	5.66	6.54	3.90	3.97	6.88	6.99	3.97	4.16	8.00	8.83	3.84	3.98
1024	7.49	9.95	5.15	5.63	9.57	10.63	5.66	5.74	11.81	13.90	5.63	5.89
2048	11.42	12.90	7.81	7.87	15.26	16.97	7.84	8.10	19.52	27.87	8.03	8.36

both NTT and INTT, illustrating that it offers significant performance benefits over the Montgomery multiplication approach, particularly as batch sizes increase. To further evaluate the impact of the branchless optimization, we compared our Plantard implementation with a conditional branching variant within the same arithmetic framework. As shown in Table III, the branchless implementation consistently exhibits faster execution times across all Kyber variants and batch sizes, highlighting the benefits of reduced warp divergence.

ACKNOWLEDGMENTS

This work was supported by the Korea NRF grant funded by the Korea government (RS-2024-00340882) and also supported by The Circle Foundation (TCF) for 1 year starting in April 2025, following the selection of the Quantum Security Research Center as a second-year recipient (2023 TCF Innovative Science Project-05).

IV. CONCLUSIONS

This letter presents an optimized Plantard arithmetic approach designed to enhance efficiency of the NTT within Kyber algorithm on GPU platforms. The proposed techniques demonstrate improvement over traditional Montgomery multiplication, achieving consistently superior throughput and reduced execution times across all variants of Kyber. Furthermore, the scalability and efficiency of the proposed approach effectively address the computational challenges faced in resource-constrained environments, thereby facilitating secure and real-time communication for IoT devices. Given that the NTT is a fundamental operation in various lattice-based schemes, such as Dilithium, the proposed optimizations hold promise for improving efficiency across multiple PQC techniques. In future, we plan to adapt these optimizations for low-end IoT devices, integrate them into broader PQC frameworks, and incorporating advanced branchless modular multiplication techniques from existing GPU implementations for higher performance.

REFERENCES

[1] J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler, and D. Stehlé, "Crystals-kyber: a cca-secure module-lattice-based kem," in *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2018, pp. 353–367.

[2] C. Zhang, D. Liu, X. Liu, X. Zou, G. Niu, B. Liu, and Q. Jiang, "Towards efficient hardware implementation of ntt for kyber on fpgas," in *2021 IEEE international symposium on circuits and systems (ISCAS)*. IEEE, 2021, pp. 1–5.

[3] Y. Xing and S. Li, "A compact hardware implementation of cca-secure key exchange mechanism crystals-kyber on fpga," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 328–356, 2021.

[4] P. Ravi, R. Poussier, S. Bhasin, and A. Chattopadhyay, "On configurable sca countermeasures against single trace attacks for the ntt: A performance evaluation study over kyber and dilithium on the arm cortex-m4," in *Security, Privacy, and Applied Cryptography Engineering: 10th International Conference, SPACE 2020, Kolkata, India, December 17–21, 2020, Proceedings 10*. Springer, 2020, pp. 123–146.

[5] L. Botros, M. J. Kannwischer, and P. Schwabe, "Memory-efficient high-speed implementation of kyber on cortex-m4," in *Progress in Cryptology–AFRICACRYPT 2019: 11th International Conference on Cryptology in Africa, Rabat, Morocco, July 9–11, 2019, Proceedings 11*. Springer, 2019, pp. 209–228.

[6] X. Ji, J. Dong, T. Deng, P. Zhang, J. Hua, and F. Xiao, "Hi-kyber: A novel high-performance implementation scheme of kyber based on gpu," *IEEE Transactions on Parallel and Distributed Systems*, 2024.

[7] N. Gupta, A. Jati, A. K. Chauhan, and A. Chattopadhyay, "Pqc acceleration using gpus: Frodokem, newhope, and kyber," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 3, pp. 575–586, 2020.

[8] W.-K. Lee and S. O. Hwang, "High throughput implementation of post-quantum key encapsulation and decapsulation on gpu for internet of things applications," *IEEE Transactions on Services Computing*, vol. 15, no. 6, pp. 3275–3288, 2021.

[9] T. Plantard, "Efficient word size modular arithmetic," *IEEE Transactions on Emerging Topics in Computing*, vol. 9, no. 3, pp. 1506–1518, 2021.

[10] J. Huang, J. Zhang, H. Zhao, Z. Liu, R. C. Cheung, Ç. K. Koç, and D. Chen, "Improved plantard arithmetic for lattice-based cryptography," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2022, no. 4, pp. 614–636, 2022.

[11] J. Huang, H. Zhao, J. Zhang, W. Dai, L. Zhou, R. C. Cheung, Ç. K. Koç, and D. Chen, "Yet another improvement of plantard arithmetic for faster kyber on low-end 32-bit iot devices," *IEEE Transactions on Information Forensics and Security*, 2024.

Muhammad Asfand Hafeez received his B.S. degree in Electrical Engineering from the University of Management and Technology in 2021. He completed his Master's degree in IT Convergence Engineering from Gachon University, South Korea, in 2024. Currently, he is pursuing his Ph.D. in Computer Science at Florida Atlantic University, USA. He is currently a researcher at the Intelligent Systems, Computer Architecture, Analytics, and Security (ISCAAS) Laboratory, and his research interests involve cryptographic engineering, GPU computing, parallel computing, and secure cryptographic protocol design.

Wai-Kong Lee received a B.Eng. in electronics and an M.Eng.Sc. from Multimedia University, Malaysia in 2006 and 2009, respectively. He received a Ph.D. in engineering from Universiti Tunku Abdul Rahman, Malaysia in 2018. Prior to joining academia, he worked in several multi-national companies, including Agilent Technologies (Malaysia) as an R&D engineer. His research interests include cryptography, numerical algorithms, GPU computing, the Internet of Things, and energy harvesting. He is currently an Associate Professor at Universiti Tunku Abdul Rahman, Malaysia.

Arslan Munir (Senior Member, IEEE) received the M.A.Sc. degree in electrical and computer engineering (ECE) from the University of British Columbia, Vancouver, Canada, in 2007, and the Ph.D. degree in ECE from the University of Florida, Gainesville, FL, USA, in 2012. From 2007 to 2008, he worked as a Software Development Engineer at the Embedded Systems Division, Mentor Graphics Corporation. He was a Postdoctoral Research Associate with the ECE Department, Rice University, Houston, TX, USA, from May 2012 to June 2014. He is currently an Associate Professor with the Department of Electrical Engineering and Computer Science, Florida Atlantic University. His current research interests include embedded and cyber-physical systems, secure and trustworthy systems, parallel computing, artificial intelligence, and computer vision. He has received many academic awards, including the Doctoral Fellowship from the Natural Sciences and Engineering Research Council (NSERC) of Canada. He earned gold medals for best performance in electrical engineering and gold medals and academic roll of honor for securing rank one in pre-engineering provincial examinations (out of approximately 300,000 candidates).

Seoung Oun Hwang received a B.S. degree in mathematics from Seoul National University, in 1993, the M.S.degree in information and communications engineering from the Pohang University of Science and Technology, in 1998, and a Ph.D. degree in computer science from the Korea Advanced Institute of Science and Technology, South Korea. He worked as a Software Engineer with LG-CNS Systems, Inc., from 1994 to 1996. He also worked as a Senior Researcher with the Electronics and Telecommunications Research Institute (ETRI), from 1998 to 2007. He worked as a Professor at the Department of Software and Communications Engineering, Hongik University, from 2008 to 2019. He is currently a Professor at the Department of Computer Engineering, at Gachon University. He is also an Editor of the ETRI Journal. His research interests include cryptography, cybersecurity, and artificial intelligence