2022 27th IEEE European Test Symposium (ETS)
- Case-Study Paper -

Super Acceleration of Dilithium in MPSoCs Critical Environments

Johanna Sepúlveda[†*] and Dominik Winkler[*]
[†]AIRBUS Defence and Space GmbH, Taufkirchen, Germany
[*]Technical University of Munich, Munich, Germany
johanna.sepulveda@airbus.com

2022 IEEE European Test Symposium (ETS) | 978-1-6654-6706-3/22/$31.00 ©2022 IEEE | DOI: 10.1109/ETS54262.2022.9810468

***Abstract*—Digital signature is a key security technology for authenticating systems and devices, thus enabling the existence of wide collaborative environments. This is also true for safety-critical systems that are constrained by strict performance requirements. Such applications are usually implemented through Multi-processors System-on-Chip (MPSoC). The dawn of quantum computing represents a threat for current cryptography, including the digital signatures. In order to prepare for such an event, electronic systems must integrate quantum-secure (post-quantum) cryptography. Dilithium is one of the main alternatives for practical implementation of post-quantum signatures. While most of the attention has been given to the security analysis and single-core software implementation, the Dilithium MPSoC exploration for high performance has been neglected. To this end, this work presents two contributions. First, the design and exploration of optimized Dilithium multi-core implementations. Second, the deployment of Dilithium on real life MPSoCs used in automotive applications and operated with a commercial RTOS. Results show that Dilithium can be efficiently implemented and optimized on a multicore architecture, improving the performance up to 48% for key generation, 34% for signature and 42% for verification when compared to single core solutions.**

***Index Terms*—Security; Post-quantum; Multi-processor; signature; Dilithium.**

I. Introduction

Signatures are one of the pillars of security, providing the means for authenticate the entities and communication parties, further securing the access control of systems and networks and avoiding the repudiation of events. Two of the most popular digital signature algorithms are based on the Rivest-Shamir-Adleman (RSA) and the elliptic curve Digital Signature (ECDSA). While these signatures are considered secure today, the foreseeable breakthrough of quantum computers represents a high risk for their security. A large quantum computer will be able to solve such hard mathematical problems in polynomial time through the execution of the Shor's quantum algorithm [1]. Thus, rendering insecure the current signatures. An attacker will be able to impersonate signers and thus mount very powerful attacks [2].

To ensure long-term communication security, Post-Quantum Cryptography (PQC) must be adopted. PQC comprises a set of algorithms that rely on mathematical problems which are secure against attacks executed on traditional and quantum computers. Moreover, PQC can be implemented using classical computing platforms. NIST is driving since 2016 the PQC standardization process, where five different security levels (from 1 to 5) have been specified. Each cryptosystem is instantiated through different parameter sets. A set of the NIST PQC finalists will be selected for the standardization. For signatures, three finalists (Dilithium, Falcon, Rainbow) were announced. Dilithium is a lattice-based PQC algorithm that relies on the hardness of the Modular Learning with Errors (MLWE) and module short integer solutions (MSIS) problems. It follows the Fiat-Shamir with aborts technique [3]. Dilithium has become one of the most promising signatures due to their strong security and very good trade-off between security and efficiency. However, signing can be a complex undertaking for many applications. In particular, for applications that require high levels of predictability, performance and that are shaped by real-time constraints, such as the automotive environment. While Dilithium has been previously deployed in different embedded platforms, there has not been any deep exploration of implementations for safety critical environments. Moreover, Multi-processor Systems on Chips (MPSoCs), widely used in such critical environments, has not been explored so as to exploit the parallelism and the pipelining in Dilithium for delivering significant speedups.

Our contribution: This work analyses Dilithium algorithm and the potential for exploiting MPSoC capabilities. We propose different implementation of Dilithium (with three different security levels). Dilithium was implemented in a MPSoc for automotive, the AURIX (Automotive Realtime Integrated NeXt Generation Architecture) with the PXROS-HR (object-oriented RTOS) developed by HighTec. PXROS-HR is certified for safety critical applications up to SIL-3 (IEC61508) and ASIL-D (ISO 26262) standards. An extensive profiling of the Dilithium PQC is provided and analyzed.

II. Related Work

Despite PQC has been deployed in multicore environments (e.g., FPGA Zynq UltraScale+MPSoC) the multicore capabilities has not been widely explored. Only two previous work have proposed PQC implementation that exploit multicore devices [4][5]. Regarding the PQC in automotive environments, PQC deployment on AURIX was proposed in [6][7][8]. All the works use bare-metal implementations and deploy two types of PQC key exchange mechanisms (NewHope, ThreeBears) on AURIX. The results of the implementation on the AURIX-TC297TF shows the feasibility of PQC and possible speed improvement of approximately 10%. Only in [7], implement a PQC signature scheme, the XMSS (extended Merkle Signature Scheme) in the AURIX TC39X. While XMSS is already specified in the IETF RFC8391 [9], it is not included in the NIST PQC standardization process. Dilithium in automotive environments was deployed for the first time in our previous seminal work in [10]. Results show the big potential for further speed up. This work explores such potential.

III. Multi-core architecture: Aurix

The widely used AURIX microcontroller from Infineon supports the standard ISO 26262 up to ASIL-D for safety-critical automotive applications such as Driver Assistance Systems (ADAS), breaking control units and steering systems.

The Infineon AURIX TC39x integrates six TriCore processors running at frequencies up to 300 MHz with 16 MB of memory flash and 6 MB of RAM. The AURIX is a Non-Uniform Memory Access (NUMA) system, where each cores

has local program and data memories. The local program memories use two Scratch Pad RAM (PSPR/DSPR) and two caches (PCACHE/DCACHE). In addition, AURIX uses two global memories: Program Memory Unit (PMU) and the Local Memory Unit (LMU). The different AURIX components are interconnected through the Shared Resource Interconnect (SRI) crossbar and the System Peripheral Bus (SPB).

The AURIX TriCore has 16-bit and 32-bit format instructions. The instruction set supports several data types, such as single precision floating-point and signed fractions. The instruction set can be used on three pipelines: the Integer Pipeline, Load/Store Pipeline and a minor pipeline for loop instructions. The three pipelines can be used simultaneously, thus achieving up to three instruction per clock cycle. The TriCore instructions can be executed in parallel and the number of clock cycles spent on data retrieval is minimal.

PXROS-HR is the RTOS used widely in automotive environments. It is fundamental to manage and monitor different applications deployed on single and multicore microcontrollers. It is certified for safety-critical applications up to SIL-3 (IEC61508) and ASIL-D (ISO 26262) standards. In PXROS-HR, all applications are encapsulated in tasks where each task possesses its protected context, data, and address space. The communication between tasks is message-based. Thereby, every task has a mailbox to which other tasks can send message objects and events to perform the data exchanges. Each message has a clear owner and authorized user to allow a transparent transfer of the address space among tasks. PXROS-HR ensures the integrity and the causality of the sent data on single and multicore configurations, granting access only to the message on the address space with MPU protection without interference at run-time.

IV. Dilithium Background and Baseline Implementation

A. Description and Background

Digital signatures are used for authenticating communication parties, to provide data integrity and to avoid the communication repudiation. The signature value is calculated based on the data and a secret key known only by the signer. Signing approaches perform three different operations: *Key generation*, *Sign* and *Verify*. First, a key pairs are generated (*key generation*). Note that keys used for encryption/decryption and signing/verifying are commonly different. The private key sk (signature key) is used together with the sign algorithm to generate a signature (*sign*). The public key pk (verification key) is used together with the verification process to check the authenticity of the message (*verify*). Since digital signature is created by private key sk of the signer, a signed transaction cannot be repudiated.

Notation All mathematical operations in Dilithium are performed in the ring $\mathcal{R}_q = \mathbb{Z}_q[x]/\ (x^n + 1)$, where n and q are both integers. Bold capital letters denote matrices of polynomials and vectors are written as bold lower case letters. The three major operations od Dilithium are described as follows.

- **Key Generation** Generates a pair of keys in three steps. First, the signer samples a matrix with $k \times l$ polynomials in the ring $R_q = \mathbb{Z}_q[X]/(X^n + 1)$ with the values $q = 2^{23} - 2^{13} + 1$ and $n = 256$.
  Second, the vectors $\mathbf{s_1}$ and $\mathbf{s_2}$ are sampled with coefficients in the range of R_q and maximum size η. Third, computes $\mathbf{t} = \mathbf{A}\mathbf{s_1} + \mathbf{s_2}$.
  The public key is $(\mathbf{A}, \mathbf{t})$ and the vectors $\mathbf{s_1}$ and $\mathbf{s_2}$ form the secret key.
- **Sign** It signs a message of 32 bytes in three steps. First, a masking vector with coefficients smaller than γ_1 is generated. Second, the signer sets $\mathbf{w_1}$ to the "high-order" bits of $\mathbf{Ay}$. The High-Bits function returns r_1 for an input r decomposed to $r = r_1\alpha + r_0$ with α being an even divisor of $q - 1$.
  Third, the potential signature $\mathbf{z}$ is computed as $\mathbf{z} = \mathbf{y} + c\mathbf{s_1}$ with the output c defined as the hash of the message and $\mathbf{w_1'}$ mapped to a polynomial consisting of $\tau \pm 1$'s and zeros, otherwise. To avoid the dependency of $\mathbf{z}$ on the secret key, the signature will be rejected if any coefficient in $\mathbf{z}$ is larger than $\gamma_1 - \beta$ or if any coefficient of the "low-order" bits of $\mathbf{Az} - c\mathbf{t}$ is larger than $\gamma_2 - \beta$. The signature is $\sigma = (\mathbf{z}, c)$.
- **Verify** It checks the validity of a signature. It computes the "high-order" bits $\mathbf{Az}$ -$c\mathbf{t}$ to $\mathbf{w_1'}$. The signature is valid if all coefficients of $\mathbf{z}$ are smaller than $\gamma_1 - \beta$ and c is the hash of the message and $\mathbf{w_1'}$.

B. Dilithium baseline implementation

The initial Dilithium implementation was obtained from the reference implementation in [3]. Baseline implementations are not optimized and are executed into a single core of the AURIX. The results are obtained after executing each algorithm 100 times and with different optimization levels, from $-$O0 to $-$O2. The clock frequency is 300 MHz. The Dilithium performance was measured with Lauterbach's TRACE32 and results are summarized in Table I.

Our previous work in [10] has shown the impact of the hash function on the Dilithium performance. It is used for the matrix generation and is responsible for 52% to 65% of the key generation execution time and of 42% to 58% of the verification execution time, depending on the security level. The NTT and modulo reduction also present a high impact on the overall Dilithium performance. Such functions are used to perform the vector-matrix multiplications. These functions are responsible of the majority of the remaining execution time after matrix generation.

V. Multicore implementation

A. Description

PXROS-HR allows locating different tasks on different cores. The functions consuming relatively large fractions of the execution time and without data dependencies are especially attractive for their outsourcing to another core (parallel execution). The function with the largest run-time in Dilithium is the matrix generation [10]. It also represents a major bottleneck, thus the acceleration of this task provides the highest potential for high performance Dilithium implementations. During the matrix generation, $K \times L$ polynomials need to be sampled. Dilithium uses different values of (*K*, *L*) for NIST security levels of 2, 3 and 5. These values are (4,4), (6,5) and (8,7), respectively. Figure 1 shows the Dilithium key generation schedule. It is composed on seven tasks. The matrix expansion is one of the bottlenecks of this operation. The public key of Dilithium only contains the seed of the matrix to reduce the storage sizes. Thus, the seed always needs to be expanded to the matrix (used in the key generation, signature and verification). During the expansion, polynomials are generated through rejection sampling of a pseudo-random input, which is the result of hashing the seed together with the indices for row and column appended. In the baseline implementation, the matrix expansion is performed through a loop that performs $K \times L$ iterations sampling, where one polynomial per iteration is generated as shown in Figure 2.

PQ Signature	Level	Flag	Key Generation			Signature			Verification		
			Min	Ave	Max	Min	Ave	Max	Min	Ave	Max
Dilithium	2		21.811	22.235	22.590	40.635	102.001	367.371	26.984	26.987	26.990
	3	00	37.784	37.794	37.806	60.655	144.542	556.447	42.150	42.154	42.157
	5		61.780	62.420	62.947	93.385	177.094	402.200	68.102	68.104	68.108
	2		6.817	6.959	7.116	12.348	26.075	98.3320	8.434	8.436	8.437
	3	01	11.814	11.809	11.822	18.462	43.937	169.326	13.169	13.170	13.171
	5		19.305	19.515	19.731	28.480	50.896	155.533	21.234	21.235	21.237
	2		5.512	5.656	5.754	10.349	23.124	121.409	7.068	7.069	7.070
	3	02	9.630	9.635	9.642	15.385	36.340	128.108	10.839	10.841	10.843
	5		15.703	15.873	16.013	23.605	46.791	170.034	17.375	17.378	17.380

TABLE I: Dilithium baseline runtimes (in milliseconds)

Fig. 1: Overview of the schedule used in the key generation

By reordering the Dilithium schedule, it is possible to sample four polynomials concurrently on four different cores. This is especially practical for security levels 2 and 5, where K is a multiple of four (4 and 8). This yields a schedule looping through L in the outer loop and the through $K/4$, while the baseline implementation loops through K on the outer level and through L on the inner loop. Figure 3 shows the restructured four cores implementation.

The 4-cores optimization is not effective for Dilithium security level 3 due to the matrix dimensions (6,5). The seed always depends on the position in the matrix where the element is sampled for. A function that loops through the lines (0 to 4) is slower on four cores than on three cores. It requires two iterations to sample six polynomials. In addition, high parallelized implementations also require a more complex management and scheduling. For Dilithium security level 3 5-cores can be used. Thereby, the loop through L can be skipped and a loop through K is able to sample five polynomials for each $i \in [0; K-1]$.

VI. Experimental Work

A. Setup Description

Dilithium signature (baseline and optimized) was deployed on AURIX TC397x operated at 300MHz. The PXROS-HR version 8.2.0 is used. The variables are stored in the DSPR

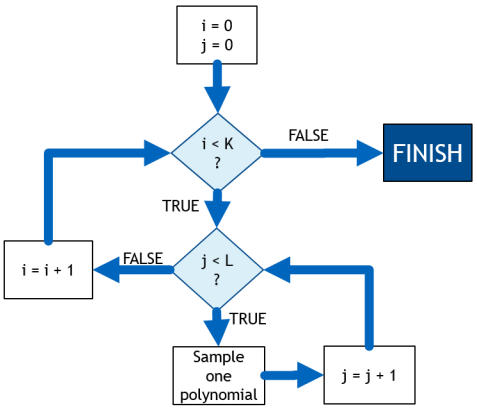

Fig. 2: Block diagram of the loop schedule to generate the matrix in the baseline implementation

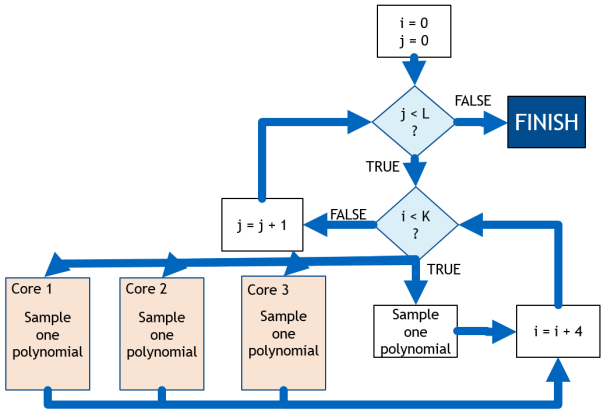

Fig. 3: The 4-core implementation: White boxes executed in Core 0; Otherwise, executed in additional cores.

Security Level	Number of Cores	Min (ms)	Ave (ms)	Max (ms)
2	1	2.971	2.971	2.972
	2	1.641	1.643	1.647
	4	0.891	0.889	0.900
3	1	5.570	5.571	5.572
	2	3.074	3.076	3.082
	3	2.140	2.142	2.149
	4	2.144	2.147	2.154
	5	1.626	1.628	1.629
5	1	10.407	10.408	10.409
	2	5.714	5.718	5.723
	4	3.064	3.067	3.075

TABLE II: Execution times (in ms) to generate the matrix on a single core (baseline) and the fastest multi-core version for each parameter set

of each core. The HighTec's toolchain with the TriCore-GCC compiler version 4.9.4.1 and the HighTec IDE are used. Three compiler optimization levels are used. The $-$O0 does not use any optimization technique. The $-$O1 is optimized to reduce the code size and execution time. The $-$O2 delivers the best performance results. The trace and debug tool TRACE32 of Lauterbach was used.

	original version			Blake3-based		
distribution	min	avg	max	min	avg	max
η	48.377	66.879	87.813	33.537	48.705	62.107
γ_1	185.070	185.243	185.717	70.257	70.853	71.947

TABLE III: Execution time (in microseconds) required to sample a polynomial for different distributions

PQ Signature	N. cores	Level	Flag	Key Generation			Signature			Verification		
				Min	Ave	Max	Min	Ave	Max	Min	Ave	Max
Dilithium baseline	1	2	O2	5.512	5.656	5.754	10.349	23.124	121.409	7.068	7.069	7.070
	1	3		9.630	9.635	9.642	15.385	36.340	128.108	10.839	10.841	10.843
	1	5		15.703	15.873	16.013	23.605	46.791	170.034	17.375	17.378	17.380
concurrent matrix sampling	4	2	O2	3.746	3.858	3.989	8.264	18.645	90.801	4.984	4.986	4.990
	3	3		6.960	6.966	6.978	11.950	28.562	103.027	7.387	7.391	7.394
	5	3		6.442	6.448	6.455	11.434	32.146	192.146	6.889	6.891	6.892
	4	5		8.890	9.063	9.241	16.260	36.436	127.544	10.043	10.048	10.054
concurrent matrix sampling & optimized vector sampling	4	2	O2	3.337	3.425	3.496	7.809	17.243	55.001	4.984	4.986	4.990
	3	3		5.779	5.786	5.796	11.364	27.177	80.245	7.385	7.387	7.390
	5	3		5.274	5.281	5.287	10.861	29.435	129.998	6.881	6.882	6.885
	4	5		8.136	8.249	8.342	15.465	29.003	93.952	10.043	10.048	10.054

TABLE IV: Dilithium run-times (ms) on AURIX TC397X and PXROS-HR with clock frequency 300 MHz

B. Results

Table II shows the matrix generation results. The sampling process on 5-cores for security level 3 is 71% faster than the original baseline implementation. The communication overhead for sampling one polynomial in a sub-task on a different core is $27.43\mu s$. For example, for Dilithium NIST security level 3, there are 6×5 polynomials. For the 5-core version, the communication overhead can be quantified as $6\times4\times27.43\mu s$. Where the 4 refers to the number of sub-tasks executed during each of the 6 iterations, while the main task (running on the main core) samples the fifth polynomial.

Additional acceleration potential for Dilithium lies in the vector sampling function. Hashing accounts for more than 90% of the run-time. Hashing can be speed up by replacing the Keccak function with the Blake3 function. This is only possible for sampling the secret vectors in the key generation and the masking vector in the signature. The functions sampling the vectors use the distribution η and γ_1. Times measured in Table III do not depend on the values of η (2 or 4) and γ_1 (2^{17} or 2^{19}).

Table IV summarizes the performance results of the baseline and the two Dilithium optimizations using O2 optimization level. The *concurrent matrix sampling* row refers to the implementation that only includes the matrix expansion optimization. The last row represents the implementation with optimized matrix and vector sampling. Results show that by using concurrent execution, an speedup up to 48%, 34% and 42% for key generation, signature and verification, respectively. In addition, by further using Blake3, a 12% and 4% speedups, when compared to the baseline, were achieved. The Blake3 optimization does not affect verification, since it does not involve vector sampling.

VII. Conclusion

This work represents the first attempt into using the multicore capabilities of embedded devices and to use an RTOS to improve the performance of the implementation of Dilithium. We present the first exploration of Dilithium in multicore environments. While the post-quantum standard is not yet defined, their evaluation in several platforms is important. Dilithium was implemented in the AURIX TC397X from Infineon Technologies AG with the Real-Time Operating System PXROS-HR from HighTec EDV-Systeme GmbH, specially designed for deployment on single and multicore microcontrollers. Single and multicore implementation were explored, revealing the potential optimization alternatives that the multicore environment has. NIST has requested the exploration of PQC to the different industrial sectors to perform different feasibility studies. This paper is an answer for such a call. We show that Dilithium can efficiently secure automotive systems and that there is an interesting potential for enhancing their deployment by using concurrent execution.

References

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

[2] D. Sikeridis, P. Kampanakis, and M. Devetsikiotis, *Post-quantum authentication in tls 1.3: A performance study*, Cryptology ePrint Archive, Report 2020/071, https://ia.cr/2020/071, 2020.

[3] L. Ducas, E. Kiltz, T. Lepoint, *et al.*, *Crystals-dilithium – algorithm specifications and supporting documentation*, https://pq-crystals.org/dilithium, Feb. 2021.

[4] J. Sepúlveda, A. Zankl, and O. Mischke, "Cache attacks and countermeasures for ntruencrypt on mpsocs: Post-quantum resistance for the iot," in *2017 30th IEEE International System-on-Chip Conference (SOCC)*, 2017, pp. 120–125.

[5] J. Sepúlveda, D. Winkler, D. Sepúlveda, *et al.*, "Post-quantum cryptography in mpsoc environments," in *2021 IFIP/IEEE 29th International Conference on Very Large Scale Integration (VLSI-SoC)*, 2021, pp. 1–6.

[6] T. Fritzmann, J. Vith, and J. Sepulveda, "Post-quantum key exchange mechanism for safety critical systems," in *17^{th} escar Europe : embedded security in cars*, 2019.

[7] J. Hermelink, T. Pöppelmann, M. Stöttinger, *et al.*, "Quantum safe authenticated key exchange protocol for automotive application," in *18^{th} escar Europe : embedded security in cars*, 2020.

[8] T. Fritzmann, J. Vith, and J. Sepulveda, "Strengthening post-quantum security for automotive systems," in *2020 23rd Euromicro Conference on Digital System Design (DSD)*, Los Alamitos, CA, USA: IEEE Computer Society, Aug. 2020, pp. 570–576.

[9] A. Huelsing, D. Butin, S.-L. Gazdag, *et al.*, *XMSS: eXtended Merkle Signature Scheme*, RFC 8391, May 2018.

[10] D. Winkler, D. Sepulveda, M. Cupelli, *et al.*, "Quantum secure high performance automotive systems," in *19^{th} escar Europe : embedded security in cars*, 2021.