



Dilithium for Memory Constrained Devices

Joppe W. Bos Joost Renes Daan Sprenkels

19 July 2022

NXP Semiconductors,
{joppe.bos,joost.renes}@nxp.com, daan@dsprenkels.com

Introduction

Memory-optimizing Dilithium

Implementation & results

Introduction

- ▶ Post-quantum signature scheme
- ▶ Based on lattices
- ▶ Performance reasonably fast: 7M cycles on Cortex-M4 [AHKS22]

Table: Dilithium key sizes in kilobytes

NIST security level	2	3	5
public key size	1.3	2.0	2.6
secret key size	2.5	4.0	4.9
signature size	2.4	3.3	4.6

More important!

**Dilithium:
winner of the NIST competition!**

Table: memory usage for Dilithium (security level 3) on Cortex-M4

publication	year	round	Sign [KiB ^a]	Verify [KiB ^a]
[GKOS18]	2018	1	84.5	53.5
[GKS21]	2021	2	9.7	9.8
PQClean [KSSW22]	2021	3	77.7	56.4
[AHKS22]	2022	3	67.4	56.6

^a 1 kibibyte is equivalent to 1024 bytes

Goal of this research:

Can we fit Dilithium in 8 KiB of RAM?

Memory-optimizing Dilithium

Algorithm Dilithium signature generation

input: secret key $(\mathbf{s}_1, \mathbf{s}_2)$; public key $(\mathbf{A}, \mathbf{t} = \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2)$; message μ

loop

$\mathbf{y} \xleftarrow{\$} S_{\gamma_1}^\ell$

$\mathbf{w}_1 := \text{HighBits}(\mathbf{A}\mathbf{y})$

$\tilde{c} := H(\mu || \mathbf{w}_1)$

$c := \text{SampleInBall}(\tilde{c})$

$\mathbf{z} := \mathbf{y} + c\mathbf{s}_1$

if $\|\mathbf{z}\|_\infty \geq \gamma_1 - \beta$ **then continue**

if $\|\text{LowBits}(\mathbf{A}\mathbf{y} - c\mathbf{s}_2)\|_\infty \geq \gamma_2 - \beta$ **then continue**

return $\sigma = (\tilde{c}, \mathbf{z})$

end loop

#1: element-wise computation & compressing of w

Algorithm Dilithium signature generation

input: secret key $(\mathbf{s}_1, \mathbf{s}_2)$; public key $(\mathbf{A}, \mathbf{t} = \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2)$; message μ

loop

$\mathbf{y} \xleftarrow{\$} S_{\gamma_1}^\ell$

$\mathbf{w}_1 := \text{HighBits}(\mathbf{A}\mathbf{y})$

$\tilde{c} := H(\mu || \mathbf{w}_1)$

$c := \text{SampleInBall}(\tilde{c})$

$\mathbf{z} := \mathbf{y} + c\mathbf{s}_1$

if $\|\mathbf{z}\|_\infty \geq \gamma_1 - \beta$ **then continue**

if $\|\text{LowBits}(\mathbf{A}\mathbf{y} - c\mathbf{s}_2)\|_\infty \geq \gamma_2 - \beta$ **then continue**

return $\sigma = (\tilde{c}, \mathbf{z})$

end loop

#1: element-wise computation & compressing of \mathbf{w}

- ▶ Compute over vectors in element-wise fashion
 - Not possible for \mathbf{w} (because overlapping lifetimes of \mathbf{w}_1 and c)
- ▶ Workaround: compress \mathbf{w}

#1: element-wise computation & compressing of \mathbf{w}

- ▶ Compute over vectors in element-wise fashion
 - Not possible for \mathbf{w} (because overlapping lifetimes of \mathbf{w}_1 and c)
- ▶ Workaround: compress \mathbf{w}
 - Every coefficient modulo $q < 2^{23}$:
 - $\Rightarrow 256 \text{ coeffs} \times 32 \text{ bits} \times \{4, 6, 8\} \text{ polynomials} = \{4.0, 6.0, 8.0\} \text{ KiB}$
 - Pack every coefficient into 24 bits:
 - $\Rightarrow 256 \text{ coeffs} \times \mathbf{24} \text{ bits} \times \{4, 6, 8\} \text{ polynomials} = \{3.0, 4.5, 6.0\} \text{ KiB}$

#2: optimizing $c \cdot s_1$ & $c \cdot s_2$

Algorithm Dilithium signature generation

input: secret key (s_1, s_2) ; public key $(\mathbf{A}, \mathbf{t} = \mathbf{A}s_1 + s_2)$; message μ

loop

$\mathbf{y} \xleftarrow{\$} S_{\gamma_1}^{\ell}$

$\mathbf{w}_1 := \text{HighBits}(\mathbf{A}\mathbf{y})$

$\tilde{c} := H(\mu || \mathbf{w}_1)$

$c := \text{SampleInBall}(\tilde{c})$

$\mathbf{z} := \mathbf{y} + c\mathbf{s}_1$

if $\|\mathbf{z}\|_{\infty} \geq \gamma_1 - \beta$ **then continue**

if $\|\text{LowBits}(\mathbf{A}\mathbf{y} - c\mathbf{s}_2)\|_{\infty} \geq \gamma_2 - \beta$ **then continue**

return $\sigma = (\tilde{c}, \mathbf{z})$

end loop

- ▶ Dilithium uses the number-theoretic transform (NTT) for multiplications
 - \triangleright Multiply $h = f \cdot g$
 - step 1: $\hat{f} := \text{NTT}(f)$
 - step 2: $\hat{g} := \text{NTT}(g)$
 - step 3: $\hat{h} = \hat{f} \circ \hat{g}$ \triangleright in-place pointwise multiplication
 - step 4: $h := \text{NTT}^{-1}(\hat{h})$
 - q is 23 bit, so need 32 bit registers for each coefficient
 - Uses 1 KiB for $f, \hat{f}, \hat{h}, \hat{h}$, plus 1 KiB for g, \hat{g}
- ▶ So multiplication needs 2 KiB (1 KiB for each operand)

#2: optimizing $c \cdot s_1$ & $c \cdot s_2$

- ▶ (Polynomial structure is $R = \mathbb{Z}_q[X]/(X^{256} + 1)$)
- ▶ $c \in R$ is small
- ▶ $s_1, s_2 \in R$ are also small

^aFor Dilithium{2,3,5}

#2: optimizing $c \cdot s_1$ & $c \cdot s_2$

- ▶ (Polynomial structure is $R = \mathbb{Z}_q[X]/(X^{256} + 1)$)
- ▶ $c \in R$ is small
- ▶ $s_1, s_2 \in R$ are also small
- ▶ \Rightarrow all coefficients x in $c \cdot s_1, c \cdot s_2 : |x| \leq \{78, 196, 120\}^a$

^aFor Dilithium{2,3,5}

#2: optimizing $c \cdot s_1$ & $c \cdot s_2$

- ▶ (Polynomial structure is $R = \mathbb{Z}_q[X]/(X^{256} + 1)$)
- ▶ $c \in R$ is small
- ▶ $s_1, s_2 \in R$ are also small
- ▶ \Rightarrow all coefficients x in $c \cdot s_1, c \cdot s_2 : |x| \leq \{78, 196, 120\}^a$
 - Don't have to use a big $q = 8380417$,
 - But can use a small $q' = \{257, 769, 257\}^a$
 - Can use 16-bit registers for coefficients (instead of 32)
 - Now we need only $0.5 \text{ KiB} + 0.5 \text{ KiB} = 1 \text{ KiB}$

^aFor Dilithium{2,3,5}

#3: optimizing $c \cdot t_0$

- ▶ Similar to $c \cdot s_1$ & $c \cdot s_2$
 - But t_0 is not small, coefficients up to $\pm 2^{13}$
 - $c \cdot t_0$ coefficients up to $\{19, 21, 20\}$ bits
 - Does not fit in 16 bits
 - So cannot use “small” (modulo- q') NTT

#3: optimizing $c \cdot t_0$

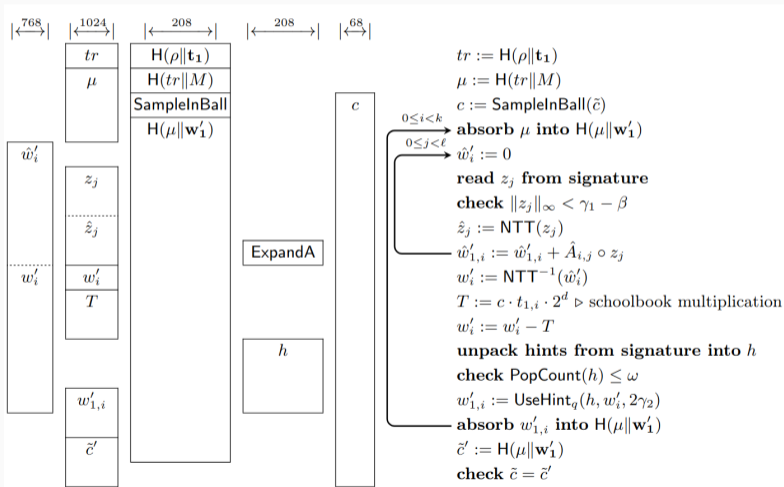
- ▶ Similar to $c \cdot s_1$ & $c \cdot s_2$
 - But t_0 is not small, coefficients up to $\pm 2^{13}$
 - $c \cdot t_0$ coefficients up to $\{19, 21, 20\}$ bits
 - Does not fit in 16 bits
 - So cannot use “small” (modulo- q') NTT
- ▶ Fall-back to schoolbook multiplication
 - Compress c into 68 bytes (68 B)
 - Unpack t_0 lazy from secret key (0 B)
 - Accumulate into product (1 KiB)

#3: optimizing $c \cdot t_0$

- ▶ Similar to $c \cdot s_1$ & $c \cdot s_2$
 - But t_0 is not small, coefficients up to $\pm 2^{13}$
 - $c \cdot t_0$ coefficients up to $\{19, 21, 20\}$ bits
 - Does not fit in 16 bits
 - So cannot use “small” (modulo- q') NTT
- ▶ Fall-back to schoolbook multiplication
 - Compress c into 68 bytes (68 B)
 - Unpack t_0 lazy from secret key (0 B)
 - Accumulate into product (1 KiB)
- ▶ Very slow, but need to do only **once**

#4: careful variable allocation

Dilithium verification:



Implementation & results

- ▶ Cross-platform (in pure C)
- ▶ No optimized assembly
- ▶ Use memory-optimization techniques
 - Generate \mathbf{A} and \mathbf{y} on-the-fly
 - Compressed format for \mathbf{w}
 - Use schoolbook multiplication for $c \cdot \mathbf{t}_0$
 - Use *small-modulus NTTs* for $c \cdot \mathbf{s}_1$ and $c \cdot \mathbf{s}_2$
 - Use optimized variable allocations

- ▶ Cross-platform (in pure C)
- ▶ No optimized assembly
- ▶ Use memory-optimization techniques
 - Generate \mathbf{A} and \mathbf{y} on-the-fly
 - Compressed format for \mathbf{w}
 - Use schoolbook multiplication for $c \cdot \mathbf{t}_0$
 - Use *small-modulus NTTs* for $c \cdot \mathbf{s}_1$ and $c \cdot \mathbf{s}_2$
 - Use optimized variable allocations
- ▶ Unfortunately not open-source

Benchmarking setup

- ▶ Integrated our implementation into pqm4 [KRSS]
- ▶ Measured memory and performance on Cortex-M4
- ▶ Expectations (at least) of memory usage [KiB]:

variant	2	3	5
K	4.3	5.8	7.3
S	4.4	5.9	7.4
V	2.2	2.2	2.2

Benchmarking setup

- ▶ Integrated our implementation into pqm4 [KRSS]
- ▶ Measured memory and performance on Cortex-M4
- ▶ Expectations (at least) of memory usage [KiB]:

variant	2	3	5
K	4.3	5.8	7.3
S	4.4	5.9	7.4
V	2.2	2.2	2.2

- ▶ Performance:
 - Expecting considerable slowdown compared to performance-optimized implementations

Table: memory usage on Cortex-M4 [KiB]

publication		Dilithium-2	Dilithium-3	Dilithium-5
[AHKS22]	S	47.9	67.4	113.3
	V	35.2	56.6	90.8
PQClean	S	50.7	77.7	— ^a
	V	35.4	56.4	— ^a
this work	S	5.0	6.5	8.1
	V	2.7	2.7	2.7

^a Did not fit on the STM32F4 board

Table: execution cycles on Cortex-M4 [kcc]^b

publication		Dilithium-2	Dilithium-3	Dilithium-5
[AHKS22]	S	4 083	6 624	8 726
	V	1 572	2 692	4 707
PQClean	S	8 034	12 987	— ^a
	V	2 223	3 666	— ^a
this work	S	18 470	36 303	44 332
	V	4 036	7 249	12 616

^a Did not fit on the STM32F4 board


^b 1 kcc is 1000 cycles

- ▶ Dilithium can be small! :)

- ▶ **Dilithium can be small! :)**
- ▶ But (compared to PQClean):
 - Approx. $2\times$ slower verification
 - Approx. $2\times - 3\times$ slower signing

- ▶ **Dilithium can be small! :)**
- ▶ But (compared to PQClean):
 - Approx. $2\times$ slower verification
 - Approx. $2\times - 3\times$ slower signing
- ▶ Especially verification (2.7 KiB / 4 Mcc) is really wonderful
 - 2.7 KiB leaves plenty of space for an OS & applications
 - 4 Mcc on a 80 MHz device is 50 ms

Questions?

-  Sedat Akleylek, Nina Bindel, Johannes A. Buchmann, Juliane Krämer, and Giorgia Azzurra Marson.


An efficient lattice-based signature scheme with provably secure instantiation.

In David Pointcheval, Abderrahmane Nitaj, and Tajjeeddine Rachidi, editors, *AFRICACRYPT 16*, volume 9646 of *LNCS*, pages 44–60. Springer, April 2016.

 Amin Abdulrahman, Vincent Hwang, Matthias J. Kannwischer, and Daan Sprenkels.

Faster Kyber and Dilithium on the Cortex-M4.

In Giuseppe Ateniese and Daniele Venturi, editors, *ACNS 2022: Applied Cryptography and Network Security*, volume 13269 of *LNCS*, pages 853–871. Springer, 2022.

 Tim Güneysu, Markus Krausz, Tobias Oder, and Julian Speith.

Evaluation of lattice-based signature schemes in embedded systems.

In *International Conference on Electronics, Circuits and Systems (ICECS)*, pages 385–388. IEEE, 2018.

-  Denisa O. C. Greconici, Matthias J. Kannwischer, and Daan Sprenkels.
Compact Dilithium implementations on Cortex-M3 and Cortex-M4.
IACR TCHES, 2021(1):1–24, 2021.
<https://tches.iacr.org/index.php/TCHES/article/view/8725>.
-  Matthias J. Kannwischer, Joost Rijneveld, Peter Schwabe, and Ko Stoffelen.
pqm4: Post-quantum crypto library for the ARM Cortex-M4.
<https://github.com/mupq/pqm4>.

 Matthias J. Kannwischer, Peter Schwabe, Douglas Stebila, and Thom Wiggers.

Improving software quality in cryptography standardization projects.

Cryptology ePrint Archive, Report 2022/337, 2022.

<https://eprint.iacr.org/2022/337>.


 Vadim Lyubashevsky, Léo Ducas, Eike Kiltz, Tancrede Lepoint, Peter Schwabe, Gregor Seiler, Damien Stehlé, and Shi Bai.

CRYSTALS-DILITHIUM.

Technical report, National Institute of Standards and Technology, 2020.

available at [https:](https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions)

[//csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions](https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions).

-  Wen Wang, Shanquan Tian, Bernhard Jungk, Nina Bindel, Patrick Longa, and Jakub Szefer.

Parameterized hardware accelerators for lattice-based cryptography.

IACR TCHES, 2020(3):269–306, 2020.

<https://tches.iacr.org/index.php/TCHES/article/view/8591>.