Faster Kyber and Dilithium on the Cortex-M4

Amin Abdulrahman[1,2] Vincent Hwang[3,4] Matthias J. Kannwischer[3] Amber Sprenkels[5]

[1]Ruhr University Bochum, Germany

[2]Max Planck Institute for Security and Privacy, Bochum, Germany

[3]Academia Sinica, Taipei, Taiwan

[4]National Taiwan University, Taipei, Taiwan

[5]Digital Security Group, Radboud University, Nijmegen, The Netherlands

23rd June 2022

Section 1

Introduction

Introduction

- Kyber, Dilithium
- Part of CRYSTALS
- NIST PQC round 3 finalists
- Lattice-based

Introduction

Kyber

- IND-CCA2 secure KEM
- Based on MLWE
- Operates on $\mathcal{R}_{3329} = \mathbb{Z}_{3329}[X]/(X^{256}+1)$
- Built to profit from NTT

Dilithium

- Signature scheme that is strongly secure under CMA
- Based on Fiat-Shamir with Aborts, MSIS, and MLWE
- Operates on $\mathcal{R}_{8380417} = \mathbb{Z}_{8380417}[X]/(X^{256}+1)$
- Built to profit from NTT

Introduction

Kyber

- IND-CCA2 secure KEM
- Based on MLWE
- Operates on $\mathcal{R}_{3329} = \mathbb{Z}_{3329}[X]/(X^{256} + 1)$
- Built to profit from NTT

Dilithium

- Signature scheme that is strongly secure under CMA
- Based on Fiat-Shamir with Aborts, MSIS, and MLWE
- Operates on $\mathcal{R}_{8380417} = \mathbb{Z}_{8380417}[X]/(X^{256} + 1)$
- Built to profit from NTT

Introduction

Kyber

- IND-CCA2 secure KEM
- Based on MLWE
- Operates on $\mathcal{R}_{3329} = \mathbb{Z}_{3329}[X]/(X^{256}+1)$
- Built to profit from NTT

Dilithium

- Signature scheme that is strongly secure under CMA
- Based on Fiat-Shamir with Aborts, MSIS, and MLWE
- Operates on $\mathcal{R}_{8380417} = \mathbb{Z}_{8380417}[X]/(X^{256}+1)$
- Built to profit from NTT

Introduction

Kyber

- IND-CCA2 secure KEM
- Based on MLWE
- Operates on $\mathcal{R}_{3329} = \mathbb{Z}_{3329}[X]/(X^{256}+1)$
- Built to profit from NTT

Dilithium

- Signature scheme that is strongly secure under CMA
- Based on Fiat-Shamir with Aborts, MSIS, and MLWE
- Operates on $\mathcal{R}_{8380417} = \mathbb{Z}_{8380417}[X]/(X^{256}+1)$
- Built to profit from NTT

Kyber

- IND-CCA2 secure KEM
- Based on MLWE
- Operates on $\mathcal{R}_{3329} = \mathbb{Z}_{3329}[X]/(X^{256}+1)$
- Built to profit from NTT

Dilithium

- Signature scheme that is strongly secure under CMA
- Based on Fiat-Shamir with Aborts, MSIS, and MLWE
- Operates on $\mathcal{R}_{8380417} = \mathbb{Z}_{8380417}[X]/(X^{256}+1)$
- Built to profit from NTT

Introduction

Kyber

- IND-CCA2 secure KEM
- Based on MLWE
- Operates on $\mathcal{R}_{3329} = \mathbb{Z}_{3329}[X]/(X^{256}+1)$
- Built to profit from NTT

Dilithium

- Signature scheme that is strongly secure under CMA
- Based on Fiat-Shamir with Aborts, MSIS, and MLWE
- Operates on $\mathcal{R}_{8380417} = \mathbb{Z}_{8380417}[X]/(X^{256}+1)$
- Built to profit from NTT

Introduction

Kyber

- IND-CCA2 secure KEM
- Based on MLWE
- Operates on $\mathcal{R}_{3329} = \mathbb{Z}_{3329}[X]/(X^{256}+1)$
- Built to profit from NTT

Dilithium

- Signature scheme that is strongly secure under CMA
- Based on Fiat-Shamir with Aborts, MSIS, and MLWE
- Operates on $\mathcal{R}_{8380417} = \mathbb{Z}_{8380417}[X]/(X^{256}+1)$
- Built to profit from NTT

Introduction

Kyber

- IND-CCA2 secure KEM
- Based on MLWE
- Operates on $\mathcal{R}_{3329} = \mathbb{Z}_{3329}[X]/(X^{256} + 1)$
- Built to profit from NTT

Dilithium

- Signature scheme that is strongly secure under CMA
- Based on Fiat-Shamir with Aborts, MSIS, and MLWE
- Operates on $\mathcal{R}_{8380417} = \mathbb{Z}_{8380417}[X]/(X^{256} + 1)$
- Built to profit from NTT

NTT in general

- Variant of the DFT defined over finite fields
- Negacyclic NTT $\triangleq$ Evaluation of polynomial at powers of primitive n-th root of unity ζ_n for $\mathcal{R}_q$ followed by twisting with powers of $2n$-th root of unity ζ_{2n}.

$$\mathtt{NTT}(a) = \hat{a} = \sum_{i=0}^{n-1} \hat{a}_i X^i \text{ with } \hat{a}_i = \sum_{j=0}^{n-1} a_j \zeta_{2n}^j \zeta_n^{ij}$$

$$\mathtt{iNTT}(\hat{a}) = a = \sum_{i=0}^{n-1} a_i X^i \text{ with } a_i = n^{-1} \zeta_{2n}^{-i} \sum_{j=0}^{n-1} \hat{a}_j \zeta_n^{-ij}$$

- Efficient NTT using Cooley–Tukey or Gentleman–Sande FFT algorithms
- Fast polynomial multiplication: Let $f, g \in \mathcal{R}_q$ and $\circ$ be base multiplication in $\mathcal{R}_q$

$$f \circ b = \mathtt{iNTT}(\mathtt{NTT}(f) \circ \mathtt{NTT}(g))$$

NTT in general

- Variant of the DFT defined over finite fields
- Negacyclic NTT $\triangleq$ Evaluation of polynomial at powers of primitive n-th root of unity ζ_n for $\mathcal{R}_q$ followed by twisting with powers of $2n$-th root of unity ζ_{2n}.

$$\mathtt{NTT}(a) = \hat{a} = \sum_{i=0}^{n-1} \hat{a}_i X^i \text{ with } \hat{a}_i = \sum_{j=0}^{n-1} a_j \zeta_{2n}^{j} \zeta_n^{ij}$$

$$\mathtt{iNTT}(\hat{a}) = a = \sum_{i=0}^{n-1} a_i X^i \text{ with } a_i = n^{-1} \zeta_{2n}^{-i} \sum_{j=0}^{n-1} \hat{a}_j \zeta_n^{-ij}$$

- Efficient NTT using Cooley–Tukey or Gentleman–Sande FFT algorithms
- Fast polynomial multiplication: Let $f, g \in \mathcal{R}_q$ and $\circ$ be base multiplication in $\mathcal{R}_q$

$$f \circ b = \mathtt{iNTT}(\mathtt{NTT}(f) \circ \mathtt{NTT}(g))$$

NTT in general

- Variant of the DFT defined over finite fields
- Negacyclic NTT $\triangleq$ Evaluation of polynomial at powers of primitive n-th root of unity ζ_n for $\mathcal{R}_q$ followed by twisting with powers of $2n$-th root of unity ζ_{2n}.

$$\mathtt{NTT}(a) = \hat{a} = \sum_{i=0}^{n-1} \hat{a}_i X^i \text{ with } \hat{a}_i = \sum_{j=0}^{n-1} a_j \zeta_{2n}^{j} \zeta_n^{ij}$$

$$\mathtt{iNTT}(\hat{a}) = a = \sum_{i=0}^{n-1} a_i X^i \text{ with } a_i = n^{-1} \zeta_{2n}^{-i} \sum_{j=0}^{n-1} \hat{a}_j \zeta_n^{-ij}$$

- Efficient NTT using Cooley–Tukey or Gentleman–Sande FFT algorithms
- Fast polynomial multiplication: Let $f, g \in \mathcal{R}_q$ and $\circ$ be base multiplication in $\mathcal{R}_q$

$$f \circ b = \mathtt{iNTT}(\mathtt{NTT}(f) \circ \mathtt{NTT}(g))$$

NTT in general

- Variant of the DFT defined over finite fields
- Negacyclic NTT $\triangleq$ Evaluation of polynomial at powers of primitive n-th root of unity ζ_n for $\mathcal{R}_q$ followed by twisting with powers of $2n$-th root of unity ζ_{2n}.

$$\mathtt{NTT}(a) = \hat{a} = \sum_{i=0}^{n-1} \hat{a}_i X^i \text{ with } \hat{a}_i = \sum_{j=0}^{n-1} a_j \zeta_{2n}^{j} \zeta_n^{ij}$$

$$\mathtt{iNTT}(\hat{a}) = a = \sum_{i=0}^{n-1} a_i X^i \text{ with } a_i = n^{-1} \zeta_{2n}^{-i} \sum_{j=0}^{n-1} \hat{a}_j {\zeta_n}^{-ij}$$

- Efficient NTT using Cooley–Tukey or Gentleman–Sande FFT algorithms
- Fast polynomial multiplication: Let $f, g \in \mathcal{R}_q$ and $\circ$ be base multiplication in $\mathcal{R}_q$

$$f \circ b = \mathtt{iNTT}(\mathtt{NTT}(f) \circ \mathtt{NTT}(g))$$

FNT

- Special case of the NTT with modulus a Fermat number $F_t := 2^{2^t} + 1$
- For F_t prime: Cyclic transformations up to $n = 2^{2^t} = F_t - 1$, negacyclic transformations up to $n = 2^{2^t - 1}$
  - $\Rightarrow$ Twiddles on first t layers are powers of two
  - $\Rightarrow$ No multiplication, only shifting
- Prime Fermat numbers: $F_0 = 3$, $F_1 = 5$, $F_2 = 17$, $F_3 = 257$, $F_4 = 65\,537$

FNT

- Special case of the NTT with modulus a Fermat number $F_t := 2^{2^t} + 1$
- For F_t prime: Cyclic transformations up to $n = 2^{2^t} = F_t - 1$, negacyclic transformations up to $n = 2^{2^t - 1}$
  - $\Rightarrow$ Twiddles on first t layers are powers of two
  - $\Rightarrow$ No multiplication, only shifting
- Prime Fermat numbers: $F_0 = 3$, $F_1 = 5$, $F_2 = 17$, $F_3 = 257$, $F_4 = 65\,537$

FNT

- Special case of the NTT with modulus a Fermat number $F_t := 2^{2^t} + 1$
- For F_t prime: Cyclic transformations up to $n = 2^{2^t} = F_t - 1$, negacyclic transformations up to $n = 2^{2^t - 1}$
  - $\Rightarrow$ Twiddles on first t layers are powers of two
  - $\Rightarrow$ No multiplication, only shifting
- Prime Fermat numbers: $F_0 = 3$, $F_1 = 5$, $F_2 = 17$, $F_3 = 257$, $F_4 = 65\,537$

FNT

- Special case of the NTT with modulus a Fermat number $F_t := 2^{2^t} + 1$
- For F_t prime: Cyclic transformations up to $n = 2^{2^t} = F_t - 1$, negacyclic transformations up to $n = 2^{2^t - 1}$
  - $\Rightarrow$ Twiddles on first t layers are powers of two
  - $\Rightarrow$ No multiplication, only shifting
- Prime Fermat numbers: $F_0 = 3$, $F_1 = 5$, $F_2 = 17$, $F_3 = 257$, $F_4 = 65\,537$

FNT

- Special case of the NTT with modulus a Fermat number $F_t := 2^{2^t} + 1$
- For F_t prime: Cyclic transformations up to $n = 2^{2^t} = F_t - 1$, negacyclic transformations up to $n = 2^{2^t - 1}$
  - $\Rightarrow$ Twiddles on first t layers are powers of two
  - $\Rightarrow$ No multiplication, only shifting
- Prime Fermat numbers: $F_0 = 3$, $F_1 = 5$, $F_2 = 17$, $F_3 = 257$, $F_4 = 65\,537$

Arm Cortex-M4

- Hardware target: STM32F407-DISCOVERY with STM32-F407VG MCU
- 1 MiB flash, 192 KiB
- Based on Armv7E-M
- 14 usable general purpose registers
- 32 single-precision floating-point registers
- Powerful DSP with useful SIMD instructions taking one cycle
  - e.g., `smul{b,t}{b,t}`, `smlad{,x}`

Arm Cortex-M4

- Hardware target: STM32F407-DISCOVERY with STM32-F407VG MCU
- 1 MiB flash, 192 KiB
- Based on Armv7E-M
- 14 usable general purpose registers
- 32 single-precision floating-point registers
- Powerful DSP with useful SIMD instructions taking one cycle
  - e.g., `smul{b,t}{b,t}`, `smlad{,x}`

Arm Cortex-M4

- Hardware target: STM32F407-DISCOVERY with STM32-F407VG MCU
- 1 MiB flash, 192 KiB
- Based on Armv7E-M
- 14 usable general purpose registers
- 32 single-precision floating-point registers
- Powerful DSP with useful SIMD instructions taking one cycle
  - e.g., `smul{b,t}{b,t}`, `smlad{,x}`

Arm Cortex-M4

- Hardware target: STM32F407-DISCOVERY with STM32-F407VG MCU
- 1 MiB flash, 192 KiB
- Based on Armv7E-M
- 14 usable general purpose registers
- 32 single-precision floating-point registers
- Powerful DSP with useful SIMD instructions taking one cycle
  - e.g., `smul{b,t}{b,t}`, `smlad{,x}`

Arm Cortex-M4

- Hardware target: STM32F407-DISCOVERY with STM32-F407VG MCU
- 1 MiB flash, 192 KiB
- Based on Armv7E-M
- 14 usable general purpose registers
- 32 single-precision floating-point registers
- Powerful DSP with useful SIMD instructions taking one cycle
  - e.g., `smul{b,t}{b,t}`, `smlad{,x}`

Arm Cortex-M4

- Hardware target: STM32F407-DISCOVERY with STM32-F407VG MCU
- 1 MiB flash, 192 KiB
- Based on Armv7E-M
- 14 usable general purpose registers
- 32 single-precision floating-point registers
- Powerful DSP with useful SIMD instructions taking one cycle
  - e.g., `smul{b,t}{b,t}`, `smlad{,x}`

Arm Cortex-M4

- Hardware target: STM32F407-DISCOVERY with STM32-F407VG MCU
- 1 MiB flash, 192 KiB
- Based on Armv7E-M
- 14 usable general purpose registers
- 32 single-precision floating-point registers
- Powerful DSP with useful SIMD instructions taking one cycle
  - e.g., `smul{b,t}{b,t}`, `smlad{,x}`

Section 2

Kyber

Kyber

- Three different parameter sets: Kyber-512, Kyber-768, and Kyber-1024.
- Same q, n for the three variants
  - $\Rightarrow$ nice for optimizing
- No $2n$-th but only n-th primitive root of unity
  - $\Rightarrow$ 7 layer incomplete NTT
  - $\Rightarrow$ 2×2 schoolbook multiplications modulo $(X^2 - \omega)$

Kyber

- Three different parameter sets: Kyber-512, Kyber-768, and Kyber-1024.
- Same q, n for the three variants
  - $\Rightarrow$ nice for optimizing
- No $2n$-th but only n-th primitive root of unity
  - $\Rightarrow$ 7 layer incomplete NTT
  - $\Rightarrow$ 2×2 schoolbook multiplications modulo $(X^2 - \omega)$

Kyber

- Three different parameter sets: Kyber-512, Kyber-768, and Kyber-1024.
- Same q, n for the three variants
  - $\Rightarrow$ nice for optimizing
- No $2n$-th but only n-th primitive root of unity
  - $\Rightarrow$ 7 layer incomplete NTT
  - $\Rightarrow$ 2×2 schoolbook multiplications modulo $(X^2 - \omega)$

Kyber

- Three different parameter sets: Kyber-512, Kyber-768, and Kyber-1024.
- Same q, n for the three variants
  - $\Rightarrow$ nice for optimizing
- No $2n$-th but only n-th primitive root of unity
  - $\Rightarrow$ 7 layer incomplete NTT
  - $\Rightarrow$ 2×2 schoolbook multiplications modulo $(X^2 - \omega)$

Kyber

- Three different parameter sets: Kyber-512, Kyber-768, and Kyber-1024.
- Same q, n for the three variants
  - $\Rightarrow$ nice for optimizing
- No $2n$-th but only n-th primitive root of unity
  - $\Rightarrow$ 7 layer incomplete NTT
  - $\Rightarrow$ 2×2 schoolbook multiplications modulo $(X^2 - \omega)$

Kyber

- Three different parameter sets: Kyber-512, Kyber-768, and Kyber-1024.
- Same q, n for the three variants
  - $\Rightarrow$ nice for optimizing
- No $2n$-th but only n-th primitive root of unity
  - $\Rightarrow$ 7 layer incomplete NTT
  - $\Rightarrow$ 2×2 schoolbook multiplications modulo $(X^2 - \omega)$

Kyber Algorithms

Algorithm: Kyber PKE key gen.

Output: public key: $pk = (\hat{\mathbf{t}}, \rho)$

Output: secret key: $sk = (\hat{\mathbf{s}})$

$\rho, \sigma \in \{0,1\}^{256} \leftarrow \texttt{sampleUniform}()$

$\hat{\mathbf{A}} \in \mathcal{R}_q^{k\times k} \leftarrow \texttt{sampleUniform}(\rho)$

$\mathbf{s}, \mathbf{e} \in \mathcal{R}_q^{k\times 1} \leftarrow \texttt{sampleCBD}^{\eta_1}(\sigma)$

$\hat{\mathbf{t}} \leftarrow \hat{\mathbf{A}} \circ \texttt{NTT}(\mathbf{s}) + \texttt{NTT}(\mathbf{e})$

return (pk, sk)

Algorithm: Kyber PKE decryption

Input : secret key: $sk = (\hat{\mathbf{s}})$

Input : compressed ciphertext: $(\mathbf{u}', v')$

Output: message $m \in \mathcal{R}_q$

$\mathbf{u} \leftarrow \texttt{Decompress}(\mathbf{u}')$

$v \leftarrow \texttt{Decompress}(v')$

return $m \leftarrow v - \texttt{iNTT}(\hat{\mathbf{s}}^T \circ \texttt{NTT}(\mathbf{u}))$

Algorithm: Kyber PKE encryption

Input : public key: $pk = (\hat{\mathbf{t}}, \rho)$

Input : message: $m \in \mathcal{R}_q$

Input : random coins: $\mu \in \{0,1\}^{256}$

Output: ciphertext $(\mathbf{u}', v')$

$\hat{\mathbf{A}} \in \mathcal{R}_q^{k\times k} \leftarrow \texttt{sampleUniform}(\rho)$

$\mathbf{r} \in \mathcal{R}_q^{k\times 1} \leftarrow \texttt{sampleCBD}^{\eta_1}(\mu)$

$\mathbf{e}_1 \in \mathcal{R}_q^{k\times 1}, e_2 \in \mathcal{R}_q \leftarrow \texttt{sampleCBD}^{\eta_2}(\mu)$

$\hat{\mathbf{r}} \leftarrow \texttt{NTT}(\mathbf{r})$

$\mathbf{u} \leftarrow \texttt{iNTT}(\hat{\mathbf{A}}^T \circ \hat{\mathbf{r}}) + \mathbf{e}_1$

$v \leftarrow \texttt{iNTT}(\hat{\mathbf{t}}^T \circ \hat{\mathbf{r}}) + e_2 + m$

return $(\texttt{Compress}(\mathbf{u}), \texttt{Compress}(v))$

Optimization: NTT and inverse NTT

- Caching in FPU registers: Store reusable values in floating point registers to avoid loading from memory
- CT-Butterflies for `iNTT`: Avoid intermediate reductions
- Better layer merging: Merge layers 7–4, 3–1 instead of 7–5, 4–2, and computing layer 1 separately

Optimization: NTT and inverse NTT

- Caching in FPU registers: Store reusable values in floating point registers to avoid loading from memory
- CT-Butterflies for `iNTT`: Avoid intermediate reductions
- Better layer merging: Merge layers 7–4, 3–1 instead of 7–5, 4–2, and computing layer 1 separately

Optimization: NTT and inverse NTT

- Caching in FPU registers: Store reusable values in floating point registers to avoid loading from memory
- CT-Butterflies for `iNTT`: Avoid intermediate reductions
- Better layer merging: Merge layers 7–4, 3–1 instead of 7–5, 4–2, and computing layer 1 separately

4 Layer Merge

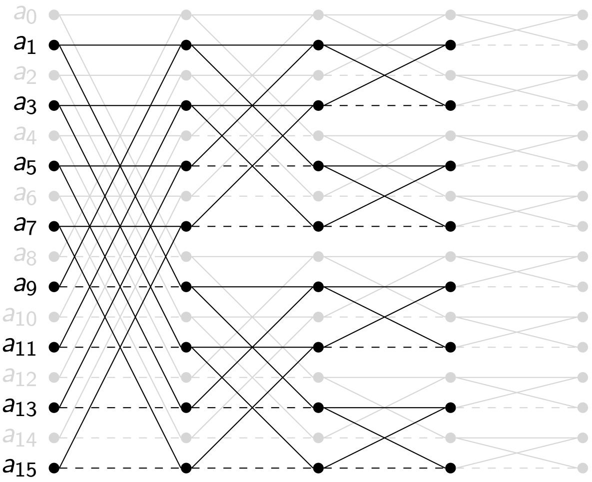

Figure: 4 layer radix 2 NTT signal flow

4 Layer Merge

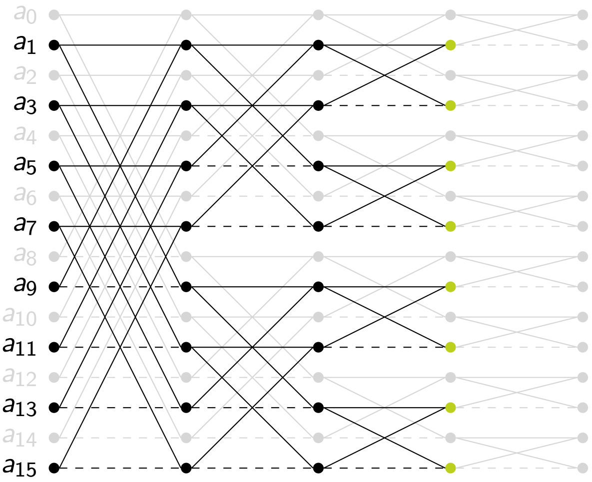

Figure: 4 layer radix 2 NTT signal flow

4 Layer Merge

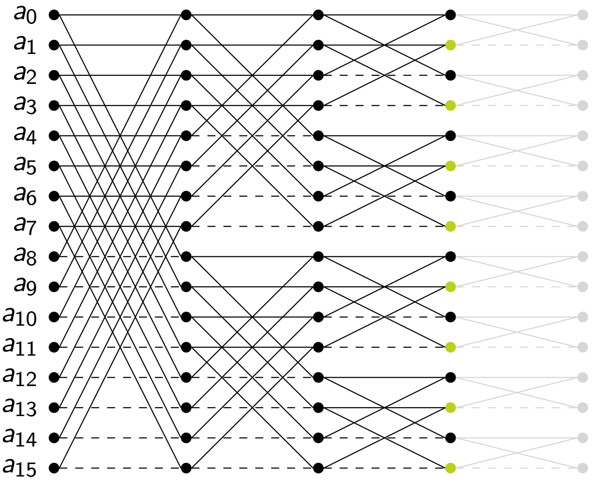

Figure: 4 layer radix 2 NTT signal flow

4 Layer Merge

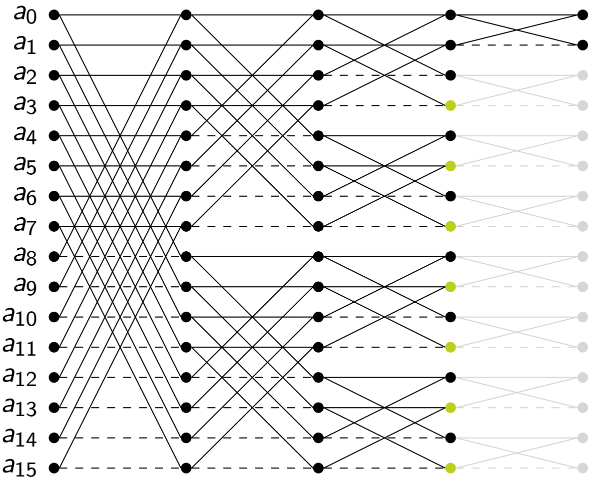

Figure: 4 layer radix 2 NTT signal flow

4 Layer Merge

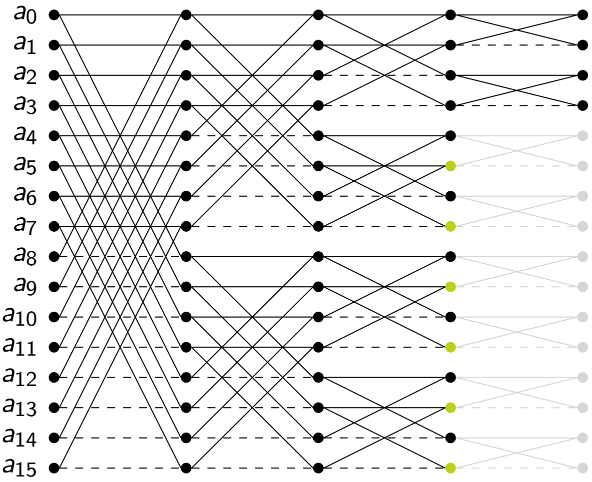

Figure: 4 layer radix 2 NTT signal flow

4 Layer Merge

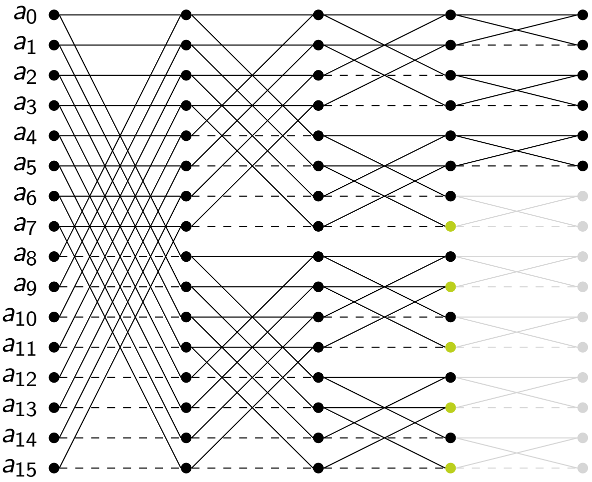

Figure: 4 layer radix 2 NTT signal flow

4 Layer Merge

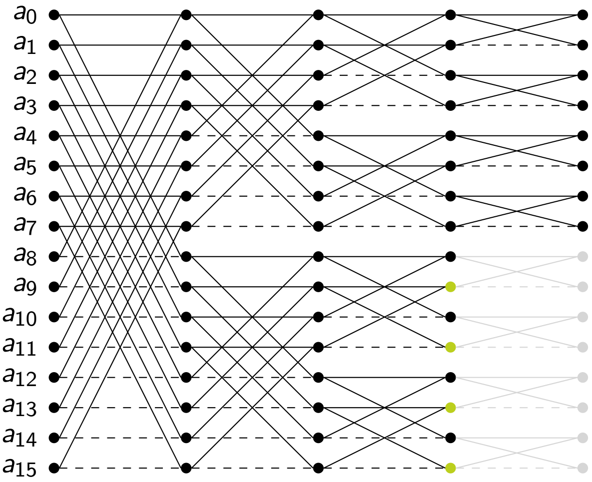

Figure: 4 layer radix 2 NTT signal flow

4 Layer Merge

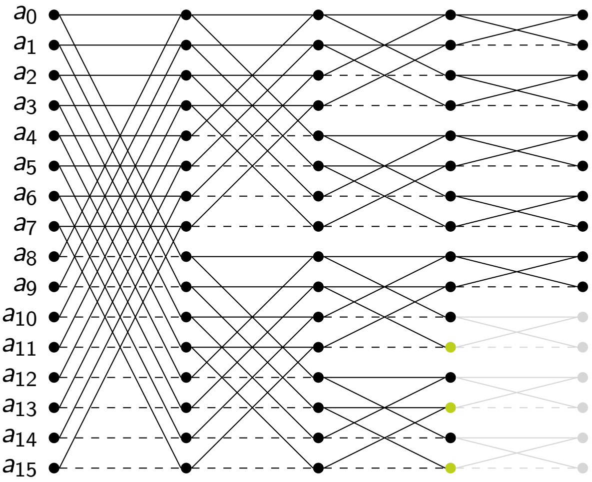

Figure: 4 layer radix 2 NTT signal flow

4 Layer Merge

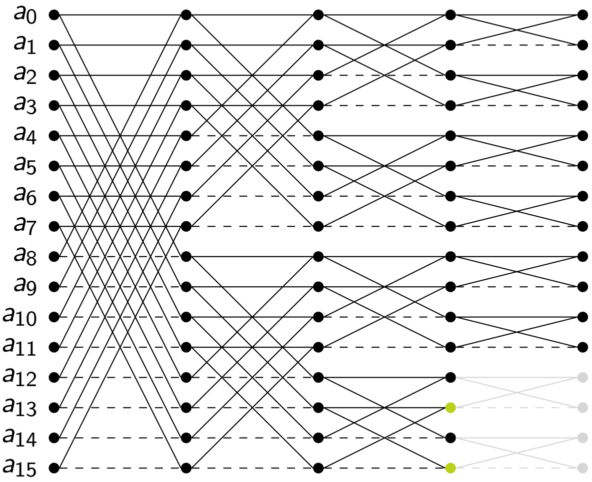

Figure: 4 layer radix 2 NTT signal flow

4 Layer Merge

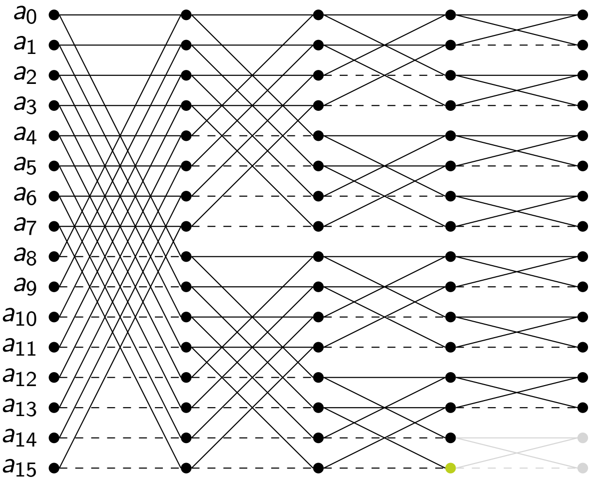

Figure: 4 layer radix 2 NTT signal flow

4 Layer Merge

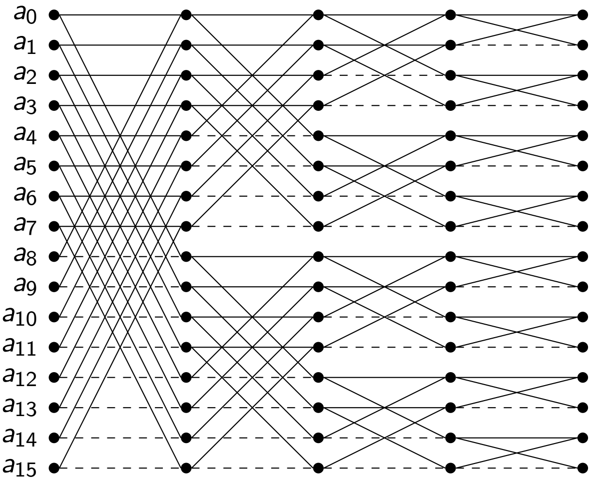

Figure: 4 layer radix 2 NTT signal flow

Kyber Algorithms

Algorithm: Kyber PKE key gen.

Output: public key: $pk = (\hat{\mathbf{t}}, \rho)$
Output: secret key: $sk = (\hat{\mathbf{s}})$

$\rho, \sigma \in \{0,1\}^{256} \leftarrow \texttt{sampleUniform}()$
$\hat{\mathbf{A}} \in \mathcal{R}_q^{k \times k} \leftarrow \texttt{sampleUniform}(\rho)$
$\mathbf{s}, \mathbf{e} \in \mathcal{R}_q^{k \times 1} \leftarrow \texttt{sampleCBD}^{\eta_1}(\sigma)$
$\hat{\mathbf{t}} \leftarrow \hat{\mathbf{A}} \circ \texttt{NTT}(\mathbf{s}) + \texttt{NTT}(\mathbf{e})$
return (pk, sk)

Algorithm: Kyber PKE decryption

Input : secret key: $sk = (\hat{\mathbf{s}})$
Input : compressed ciphertext: $(\mathbf{u}', v')$
Output: message $m \in \mathcal{R}_q$

$\mathbf{u} \leftarrow \texttt{Decompress}(\mathbf{u}')$
$v \leftarrow \texttt{Decompress}(v')$
return $m \leftarrow v - \texttt{iNTT}(\hat{\mathbf{s}}^T \circ \texttt{NTT}(\mathbf{u}))$

Algorithm: Kyber PKE encryption

Input : public key: $pk = (\hat{\mathbf{t}}, \rho)$
Input : message: $m \in \mathcal{R}_q$
Input : random coins: $\mu \in \{0,1\}^{256}$
Output: ciphertext $(\mathbf{u}', v')$

$\hat{\mathbf{A}} \in \mathcal{R}_q^{k \times k} \leftarrow \texttt{sampleUniform}(\rho)$
$\mathbf{r} \in \mathcal{R}_q^{k \times 1} \leftarrow \texttt{sampleCBD}^{\eta_1}(\mu)$
$\mathbf{e}_1 \in \mathcal{R}_q^{k \times 1}, e_2 \in \mathcal{R}_q \leftarrow \texttt{sampleCBD}^{\eta_2}(\mu)$
$\hat{\mathbf{r}} \leftarrow \texttt{NTT}(\mathbf{r})$
$\mathbf{u} \leftarrow \texttt{iNTT}(\hat{\mathbf{A}}^T \circ \hat{\mathbf{r}}) + \mathbf{e}_1$
$v \leftarrow \texttt{iNTT}(\hat{\mathbf{t}}^T \circ \hat{\mathbf{r}}) + e_2 + m$
return $(\texttt{Compress}(\mathbf{u}), \texttt{Compress}(v))$

Optimization: Barrett Reduction

Algorithm: Packed Barrett Reduction [BKS19]

Input : $a = (a_t \mid\mid a_b)$
Output: $c = (c_t \mid\mid c_b) \bmod^{\pm} q$

smulbb $t_0, a, \lfloor \frac{2^{26}}{q} \rceil$
smultb $t_1, a, \lfloor \frac{2^{26}}{q} \rceil$
asr $t_0, t_0, \#26$
asr $t_1, t_1, \#26$
smulbb t_0, t_0, q
smulbb t_1, t_1, q
pkhbt $t_0, t_0, t_1,$ lsl #16
usub16 r, a, t_0

Algorithm: Improved Packed Barrett Reduction

Input : $a = (a_t \mid\mid a_b)$
Output: $c = (c_t \mid\mid c_b) \bmod^{\pm} q$

smlawb $t_0, -\lfloor \frac{2^{32}}{q} \rceil, a, 2^{15}$
smlabt t_0, q, t_0, a
smlawt $t_1, -\lfloor \frac{2^{32}}{q} \rceil, a, 2^{15}$
smulbt t_1, q, t_1
add $t_1, a, t_1,$ lsl #16
pkhbt $c, t_0, t_1,$ lsl #16

- Note: Output range not in $[0, q)$ but $[-\frac{q-1}{2}, \frac{q-1}{2}]$ for odd q

Optimization: Matrix-vector and inner product

Optimization based on technique presented in [Bec+21]:

- Recall base multiplication for Kyber: Let $\hat{a} = \hat{\mathbf{A}}_{m,n}, \hat{s} = \hat{\mathbf{s}}_m$. For $\hat{c} = \hat{a} \circ \hat{s}$

$$\hat{c}_{2i} + \hat{c}_{2i+1}X = (\hat{a}_{2i} + \hat{a}_{2i+1}X)(\hat{s}_{2i} + \hat{s}_{2i+1}X) \bmod (X^2 - \zeta^{2\mathrm{br}_7(i)+1}), \text{ with}$$

$$\hat{c}_{2i} = \hat{a}_{2i}\hat{s}_{2i} + \hat{a}_{2i+1}\hat{s}_{2i+1}\zeta^{2\mathrm{br}_7(i)+1}$$

$$\hat{c}_{2i+1} = \hat{a}_{2i}\hat{s}_{2i+1} + \hat{s}_{2i}\hat{a}_{2i+1}$$

- Notice:

$$\begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix} \cdot \begin{bmatrix} \cdot \\ \cdot \\ \cdot \end{bmatrix} = \begin{bmatrix} \cdot \\ \cdot \\ \cdot \end{bmatrix}$$

- Asymmetric multiplication: Cache/Reuse multiplication with twiddle factors during base multiplication.

Optimization: Matrix-vector and inner product

Optimization based on technique presented in [Bec+21]:

- Recall base multiplication for Kyber: Let $\hat{a} = \hat{\mathbf{A}}_{m,n}, \hat{s} = \hat{\mathbf{s}}_m$. For $\hat{c} = \hat{a} \circ \hat{s}$

$$\hat{c}_{2i} + \hat{c}_{2i+1}X = (\hat{a}_{2i} + \hat{a}_{2i+1}X)(\hat{s}_{2i} + \hat{s}_{2i+1}X) \bmod (X^2 - \zeta^{2\mathrm{br}_7(i)+1}), \text{ with}$$

$$\hat{c}_{2i} = \hat{a}_{2i}\hat{s}_{2i} + \hat{a}_{2i+1}\hat{s}_{2i+1}\zeta^{2\mathrm{br}_7(i)+1}$$

$$\hat{c}_{2i+1} = \hat{a}_{2i}\hat{s}_{2i+1} + \hat{s}_{2i}\hat{a}_{2i+1}$$

- Notice:

$$\begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix} \cdot \begin{bmatrix} \cdot \\ \cdot \\ \cdot \end{bmatrix} = \begin{bmatrix} \cdot \\ \cdot \\ \cdot \end{bmatrix}$$

- Asymmetric multiplication: Cache/Reuse multiplication with twiddle factors during base multiplication.

Optimization: Matrix-vector and inner product

Optimization based on technique presented in [Bec+21]:

- Recall base multiplication for Kyber: Let $\hat{a} = \hat{\mathbf{A}}_{m,n}, \hat{s} = \hat{\mathbf{s}}_m$. For $\hat{c} = \hat{a} \circ \hat{s}$

$$\hat{c}_{2i} + \hat{c}_{2i+1}X = (\hat{a}_{2i} + \hat{a}_{2i+1}X)(\hat{s}_{2i} + \hat{s}_{2i+1}X) \bmod (X^2 - \zeta^{2\mathrm{br}_7(i)+1}), \text{ with}$$

$$\hat{c}_{2i} = \hat{a}_{2i}\hat{s}_{2i} + \hat{a}_{2i+1}\hat{s}_{2i+1}\zeta^{2\mathrm{br}_7(i)+1}$$

$$\hat{c}_{2i+1} = \hat{a}_{2i}\hat{s}_{2i+1} + \hat{s}_{2i}\hat{a}_{2i+1}$$

- Notice:

$$\begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix} \cdot \begin{bmatrix} \cdot \\ \cdot \\ \cdot \end{bmatrix} = \begin{bmatrix} \cdot \\ \cdot \\ \cdot \end{bmatrix}$$

- Asymmetric multiplication: Cache/Reuse multiplication with twiddle factors during base multiplication.

Optimization: Matrix-vector and inner product

Optimization based on technique presented in [Bec+21]:

- Recall base multiplication for Kyber: Let $\hat{a} = \hat{\mathbf{A}}_{m,n}, \hat{s} = \hat{\mathbf{s}}_m$. For $\hat{c} = \hat{a} \circ \hat{s}$

$$\hat{c}_{2i} + \hat{c}_{2i+1}X = (\hat{a}_{2i} + \hat{a}_{2i+1}X)(\hat{s}_{2i} + \hat{s}_{2i+1}X) \bmod (X^2 - \zeta^{2\mathrm{br}_7(i)+1}), \text{ with}$$

$$\hat{c}_{2i} = \hat{a}_{2i}\hat{s}_{2i} + \hat{a}_{2i+1}\hat{s}_{2i+1}\zeta^{2\mathrm{br}_7(i)+1}$$

$$\hat{c}_{2i+1} = \hat{a}_{2i}\hat{s}_{2i+1} + \hat{s}_{2i}\hat{a}_{2i+1}$$

- Notice:

$$\begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix} \cdot \begin{bmatrix} \cdot \\ \cdot \\ \cdot \end{bmatrix} = \begin{bmatrix} \cdot \\ \cdot \\ \cdot \end{bmatrix}$$

- Asymmetric multiplication: Cache/Reuse multiplication with twiddle factors during base multiplication.

Kyber Algorithms

Algorithm: Kyber PKE key gen.

Output: public key: $pk = (\hat{\mathbf{t}}, \rho)$
Output: secret key: $sk = (\hat{\mathbf{s}})$

$\rho, \sigma \in \{0,1\}^{256} \leftarrow \texttt{sampleUniform}()$
$\hat{\mathbf{A}} \in \mathcal{R}_q^{k \times k} \leftarrow \texttt{sampleUniform}(\rho)$
$\mathbf{s}, \mathbf{e} \in \mathcal{R}_q^{k \times 1} \leftarrow \texttt{sampleCBD}^{\eta_1}(\sigma)$
$\hat{\mathbf{t}} \leftarrow \hat{\mathbf{A}} \circ \texttt{NTT}(\mathbf{s}) + \texttt{NTT}(\mathbf{e})$
return (pk, sk)

Algorithm: Kyber PKE encryption

Input : public key: $pk = (\hat{\mathbf{t}}, \rho)$
Input : message: $m \in \mathcal{R}_q$
Input : random coins: $\mu \in \{0,1\}^{256}$
Output: ciphertext $(\mathbf{u}', v')$

$\hat{\mathbf{A}} \in \mathcal{R}_q^{k \times k} \leftarrow \texttt{sampleUniform}(\rho)$
$\mathbf{r} \in \mathcal{R}_q^{k \times 1} \leftarrow \texttt{sampleCBD}^{\eta_1}(\mu)$
$\mathbf{e}_1 \in \mathcal{R}_q^{k \times 1}, e_2 \in \mathcal{R}_q \leftarrow \texttt{sampleCBD}^{\eta_2}(\mu)$
$\hat{\mathbf{r}} \leftarrow \texttt{NTT}(\mathbf{r})$
$\mathbf{u} \leftarrow \texttt{iNTT}(\hat{\mathbf{A}}^T \circ \hat{\mathbf{r}}) + \mathbf{e}_1$
$v \leftarrow \texttt{iNTT}(\hat{\mathbf{t}}^T \circ \hat{\mathbf{r}}) + e_2 + m$
return $(\texttt{Compress}(\mathbf{u}), \texttt{Compress}(v))$

Kyber Algorithms

Algorithm: Kyber PKE key gen.

Output: public key: $pk = (\hat{\mathbf{t}}, \rho)$
Output: secret key: $sk = (\hat{\mathbf{s}})$

$\rho, \sigma \in \{0,1\}^{256} \leftarrow \texttt{sampleUniform}()$
$\hat{\mathbf{A}} \in \mathcal{R}_q^{k\times k} \leftarrow \texttt{sampleUniform}(\rho)$
$\mathbf{s}, \mathbf{e} \in \mathcal{R}_q^{k\times 1} \leftarrow \texttt{sampleCBD}^{\eta_1}(\sigma)$
$\hat{\mathbf{t}} \leftarrow \hat{\mathbf{A}} \circ \texttt{NTT}(\mathbf{s}) + \texttt{NTT}(\mathbf{e})$
return (pk, sk)

Algorithm: Kyber PKE encryption

Input : public key: $pk = (\hat{\mathbf{t}}, \rho)$
Input : message: $m \in \mathcal{R}_q$
Input : random coins: $\mu \in \{0,1\}^{256}$
Output: ciphertext $(\mathbf{u}', v')$

$\hat{\mathbf{A}} \in \mathcal{R}_q^{k\times k} \leftarrow \texttt{sampleUniform}(\rho)$
$\mathbf{r} \in \mathcal{R}_q^{k\times 1} \leftarrow \texttt{sampleCBD}^{\eta_1}(\mu)$
$\mathbf{e}_1 \in \mathcal{R}_q^{k\times 1}, e_2 \in \mathcal{R}_q \leftarrow \texttt{sampleCBD}^{\eta_2}(\mu)$
$\hat{\mathbf{r}} \leftarrow \texttt{NTT}(\mathbf{r})$
$\mathbf{u} \leftarrow \texttt{iNTT}(\hat{\mathbf{A}}^T \circ \hat{\mathbf{r}}) + \mathbf{e}_1$
$v \leftarrow \texttt{iNTT}(\hat{\mathbf{t}}^T \circ \hat{\mathbf{r}}) + e_2 + m$
return $(\texttt{Compress}(\mathbf{u}), \texttt{Compress}(v))$

Optimization: Matrix-vector and inner product

Better accumulation based on [Chu+21]:

- Kyber's small prime allows for accumulation without intermediate reductions in the matrix-vector product.

Section 3

Dilithium

Dilithium

- Three different parameter sets: Dilithium2, Dilithium3, Dilithium5

Table: Overview of Dilithium's parameter sets [Bai+20]

Scheme	NIST level	(k,l)	η	τ	γ_1	γ_2	#reps	$\lvert pk \rvert$	$\lvert sig \rvert$
Dilithium2	2	$(4,4)$	2	39	2^{17}	$(q-1)/88$	4.25	1312 B	2420 B
Dilithium3	3	$(6,5)$	4	49	2^{19}	$(q-1)/32$	5.1	1952 B	3293 B
Dilithium5	5	$(8,7)$	2	60	2^{19}	$(q-1)/32$	3.85	2592 B	4595 B

- Same q, n for the three variants
  - $\Rightarrow$ nice for optimizing
- In contrast to Kyber, $2n$-th primitive root of unity exists
  - $\Rightarrow$ Complete NTT

Dilithium Algorithms

Algorithm: Dilithium key generation

Output: secret key $sk = (\rho, K, tr, \mathbf{s}_1, \mathbf{s}_2, \mathbf{t}_0)$

Output: public key $pk = (\rho, \mathbf{t}_1)$

$\rho, \varsigma, K \in \{0,1\}^{256} \leftarrow$ `sampleUniform()`;

$\mathbf{s}_1 \in [-\eta, \eta]^{l \times 1}, \mathbf{s}_2 \in [-\eta, \eta]^{k \times 1} \leftarrow$ `sampleUniform`(ς) ;

$\hat{\mathbf{A}} \in \mathcal{R}_q^{k \times l} \leftarrow \mathtt{ExpandA}(\rho)$;

$\mathbf{t} \leftarrow \mathtt{iNTT}(\hat{\mathbf{A}} \circ \mathtt{NTT}(\mathbf{s}_1)) + \mathbf{s}_2$;

$(\mathbf{t}_1, \mathbf{t}_0) \leftarrow \mathtt{Power2Round}(\mathbf{t})$;

$tr \in \{0,1\}^{256} \leftarrow \mathtt{CRH}(\rho \| \mathbf{t}_1)$;

return (pk, sk)

Algorithm: Dilithium verification

Input : public key $pk = (\rho, \mathbf{t}_1)$

Input : message: $M \in \{0,1\}^*$

Input : signature $\sigma = (\mathbf{z}, \mathbf{h}, \tilde{c})$

Output: signature valid or signature invalid

$\hat{\mathbf{A}} \in \mathcal{R}_q^{k \times l} \leftarrow \mathtt{ExpandA}(\rho)$;

$c \leftarrow \mathtt{SampleInBall}(\tilde{c})$;

$\mu \in \{0,1\}^{512} \leftarrow \mathtt{CRH}(\mathtt{CRH}(\rho \| \mathbf{t}_1) \| M)$;

$\mathbf{w}'_1 \leftarrow \mathtt{UseHint}(\mathbf{h}, \mathtt{iNTT}(\hat{\mathbf{A}} \circ \mathtt{NTT}(\mathbf{z}) - \mathtt{NTT}(c) \circ \mathtt{NTT}(2^d \cdot \mathbf{t}_1)))$;

if $\|\mathbf{z}\|_\infty < \gamma_1 - \beta$ and $\tilde{c} = \mathtt{H}(\mu \| \mathbf{w}'_1)$ and # of 1's in $\mathbf{h} \leq \omega$ **then**

  return signature valid;

else

  return signature invalid;

Dilithium Algorithms

Algorithm: Dilithium signing

Input : secret key $sk = (\rho, K, tr, \mathbf{s}_1, \mathbf{s}_2, \mathbf{t}_0)$
Input : message: $M \in \{0,1\}^*$
Output: signature $\sigma = (\mathbf{z}, \mathbf{h}, \tilde{c})$

$\hat{\mathbf{A}} \in \mathcal{R}_q^{k \times l} \leftarrow \mathtt{ExpandA}(\rho)$;
$\mu \in \{0,1\}^{512} \leftarrow \mathtt{CRH}(tr \| M)$;
$\kappa \leftarrow 0, (\mathbf{z}, \mathbf{h}) \leftarrow \perp$;
$\rho' \in \{0,1\}^{512} \leftarrow \mathtt{CRH}(K \| \mu)$;
$\hat{\mathbf{s}}_1 \leftarrow \mathtt{NTT}(\mathbf{s}_1)$, $\hat{\mathbf{s}}_2 \leftarrow \mathtt{NTT}(\mathbf{s}_2)$, $\hat{\mathbf{t}}_0 := \mathtt{NTT}(\mathbf{t}_0)$;
while $(\mathbf{z}, \mathbf{h}) = \perp$ **do**
  $\mathbf{y} \in \mathcal{R}_q^{l \times 1} \leftarrow \mathtt{ExpandMask}(\rho', \kappa)$;
  $\mathbf{w} \leftarrow \mathtt{iNTT}(\hat{\mathbf{A}} \circ \mathtt{NTT}(\mathbf{y}))$;
  $\mathbf{w}_1 \leftarrow \mathtt{HighBits}(\mathbf{w}, 2\gamma_2)$;
  $\tilde{c} \leftarrow \mathtt{H}(\mu \| \mathbf{w}_1$;
  $c \leftarrow \mathtt{SampleInBall}(\tilde{c})$;
  $\hat{c} \leftarrow \mathtt{NTT}(c)$;
  $\mathbf{z} \leftarrow \mathbf{y} + \mathtt{iNTT}(\hat{c} \circ \hat{\mathbf{s}}_1)$;
  $\mathbf{r}_0 \leftarrow \mathtt{LowBits}(\mathbf{w} - \mathtt{iNTT}(\hat{c} \circ \hat{\mathbf{s}}_2), 2\gamma_2)$;
  if $\|\mathbf{z}\|_\infty \geq \gamma_1 - \beta$ or $\|\mathbf{r}_0\|_\infty \geq \gamma_2 - \beta$ **then**
    $(\mathbf{z}, \mathbf{h}) \leftarrow \perp$
  else
    $\mathbf{h} \leftarrow \mathtt{MakeHint}(-\mathtt{iNTT}(\hat{c} \circ \hat{\mathbf{t}}_0), \mathbf{w} -$
    $\mathtt{iNTT}(\hat{c} \circ \hat{\mathbf{s}}_2 + \mathtt{iNTT}(\hat{c} \circ \hat{\mathbf{t}}_0)), 2\gamma_2)$;
    if $\|\mathtt{iNTT}(\hat{c} \circ \hat{\mathbf{t}}_0)\|_\infty \geq \gamma_2$ or $\#$ of 1's in
    $\mathbf{h} > \omega$ **then**
      $(\mathbf{z}, \mathbf{h}) \leftarrow \perp$
  $\kappa \leftarrow \kappa + l$;
return $\sigma = (\mathbf{z}, \mathbf{h}, \tilde{c})$

Optimization: NTTs

Similar techniques as for Kyber:

- Better layer merging: Merge layers 7–5, 4–2, 1–0, instead of 7–6, 5–4, 3–2, 1–0.
- CT-Butterflies for `iNTT`

Optimization: NTTs

Similar techniques as for Kyber:

- Better layer merging: Merge layers 7–5, 4–2, 1–0, instead of 7–6, 5–4, 3–2, 1–0.
- CT-Butterflies for `iNTT`

Optimization: Small NTTs

- Recall: c consists of τ-many ± 1s, $\boldsymbol{s}_1, \boldsymbol{s}_2$ have elements in $[-\eta, \eta]$
  - $\Rightarrow$ $c\boldsymbol{s}_1$ and $c\boldsymbol{s}_2$ bounded by $\tau\eta$
  - $\Rightarrow$ Regard computation as in $\mathbb{Z}_{q'}$ with $q' > 2\tau\eta$ [Chu+21]
- Some freedom for choosing q'

Table: Choosing q'

Scheme	η	τ	$2\tau\eta$	q'
Dilithium2	2	39	156	$F_3 = 257$
Dilithium3	4	49	392	769
Dilithium5	2	60	240	$F_3 = 257$

Optimization: Small NTTs

- Recall: c consists of τ-many ± 1s, $\boldsymbol{s}_1, \boldsymbol{s}_2$ have elements in $[-\eta, \eta]$
  - $\Rightarrow$ $c\boldsymbol{s}_1$ and $c\boldsymbol{s}_2$ bounded by $\tau\eta$
  - $\Rightarrow$ Regard computation as in $\mathbb{Z}_{q'}$ with $q' > 2\tau\eta$ [Chu+21]
- Some freedom for choosing q'

Table: Choosing q'

Scheme	η	τ	$2\tau\eta$	q'
Dilithium2	2	39	156	$F_3 = 257$
Dilithium3	4	49	392	769
Dilithium5	2	60	240	$F_3 = 257$

Optimization: Small NTTs

- Recall: c consists of τ-many ± 1s, $\boldsymbol{s}_1, \boldsymbol{s}_2$ have elements in $[-\eta, \eta]$
  - $\Rightarrow$ $c\boldsymbol{s}_1$ and $c\boldsymbol{s}_2$ bounded by $\tau\eta$
  - $\Rightarrow$ Regard computation as in $\mathbb{Z}_{q'}$ with $q' > 2\tau\eta$ [Chu+21]
- Some freedom for choosing q'

Table: Choosing q'

Scheme	η	τ	$2\tau\eta$	q'
Dilithium2	2	39	156	$F_3 = 257$
Dilithium3	4	49	392	769
Dilithium5	2	60	240	$F_3 = 257$

Optimization: Small NTTs

- Recall: c consists of τ-many ± 1s, $\boldsymbol{s}_1, \boldsymbol{s}_2$ have elements in $[-\eta, \eta]$
  - $\Rightarrow$ $c\boldsymbol{s}_1$ and $c\boldsymbol{s}_2$ bounded by $\tau\eta$
  - $\Rightarrow$ Regard computation as in $\mathbb{Z}_{q'}$ with $q' > 2\tau\eta$ [Chu+21]
- Some freedom for choosing q'

Table: Choosing q'

Scheme	η	τ	$2\tau\eta$	q'
Dilithium2	2	39	156	$F_3 = 257$
Dilithium3	4	49	392	769
Dilithium5	2	60	240	$F_3 = 257$

FNT for Dilithium2 and Dilithium3

- CT-Butterfly: $(a, b) \mapsto (a + \omega b, a - \omega b)$ can be implemented with `mla` and `mls`
- First $t = 3$ layers have power of two twiddle factor

  $\Rightarrow$ Efficient implementation without loading using barrel shifter and $\log \omega$ as twiddle factor

Algorithm: $\texttt{CT\_FNT}(a, b, \texttt{logW})$.

Input : $(\mathrm{a}, \mathrm{b}) = (a, b)$

Output: $(\mathrm{a}, \mathrm{b}) = (a + 2^{\texttt{logW}} b, a - 2^{\texttt{logW}} b)$

```
add a, a, b, lsl #logW;
sub b, a, b, lsl #(logW+1);
```

Small NTT for Dilithium3

- Incompatible with FNT over F_3
- Kyber NTT/iNTT with $q' = 769$ and most reductions left out
- Experiments with $q' = F_4 = 65537$ yielding no speed-up over $q' = 769$
- $q' = 3329$ also possible for code re-use

Section 4

Results

Benchmarking Setup

- Based on pqm4
- Clock reduced to 24 MHz
- `arm-none-eabi-gcc` version 10.2.1 with `-O3`
- Keccak from pqm4
- Randomness from hardware RNG

Benchmarking Setup

- Based on pqm4
- Clock reduced to 24 MHz
- `arm-none-eabi-gcc` version 10.2.1 with `-O3`
- Keccak from pqm4
- Randomness from hardware RNG

Benchmarking Setup

- Based on pqm4
- Clock reduced to 24 MHz
- `arm-none-eabi-gcc` version 10.2.1 with `-O3`
- Keccak from pqm4
- Randomness from hardware RNG

Benchmarking Setup

- Based on pqm4
- Clock reduced to 24 MHz
- `arm-none-eabi-gcc` version 10.2.1 with `-O3`
- Keccak from pqm4
- Randomness from hardware RNG

Benchmarking Setup

- Based on pqm4
- Clock reduced to 24 MHz
- `arm-none-eabi-gcc` version 10.2.1 with -O3
- Keccak from pqm4
- Randomness from hardware RNG

NTT related functions

Table: Cycle counts for transformation operations of Kyber and Dilithium. `NTT` and `iNTT` correspond to the schemes default transformations, i.e., $q = 3329$ for Kyber and $q = 8380417$ for Dilithium. The `NTT` with $q = 257$ is deployed for Dilithium2 and Dilithium5, and the `NTT` with $q = 769$ is used used for Dilithium3.

	Prime	Implementation	`NTT`	`iNTT`	basemul
Kyber	$q = 3329$	[Alk+20]	6 852	6 979	2 317
		This work	5 992	5 491/6 282[a]	1 613[b]
Dilithium	$q = 8380417$	[GKS20]	8 540	8 923	1 955
		This work	8 093	8 415	1 955
	$q = 257$	This work	5 524	5 563	1 225
	$q = 769$	[Abd+21] (6-layer)	4 852	4 817	2 966
		This work	5 200	5 537	1 740

[a] First value is for speed-optimization, second for stack-optimization.
[b] Asymmetric basemul as used in the IP (enc). As the basemul in the MVP and IP consists of individual function calls, the cycle count is not straight forward to measure.

Kyber: Matrix-vector and inner product

Table: Cycle counts for matrix-vector and inner products used in Kyber.

implementation	variant	operation	Kyber-512	Kyber-768	Kyber-1024
pqm4		Matrix-Vector Product[a]	66 291	127 634	209 517
		Matrix-Vector Product[b]	226 580	484 077	840 498
		Inner Product (enc)	11 978	14 696	17 429
		Inner Product (dec)	29 888	41 910	53 792
This work	speed	Matrix-Vector Product[a]	55 746	106 380	172 152
		Matrix-Vector Product[b]	211 606	457 213	796 349
		Inner Product (enc)	8 762	10 331	11 898
		Inner Product (dec)	23 425	32 354	41 275
	stack	Matrix-Vector Product[a]	58 028	112 503	184 149
		Matrix-Vector Product[b]	214 053	463 590	808 206
		Inner Product (enc)	11 218	13 877	16 733
		Inner Product (dec)	24 722	34 167	43 619

[a] Measurement excluding the hashing.
[b] Measurement including the hashing.

Kyber: Scheme performance

Table: Cycle counts and stack usage for Kyber for the key generation, encapsulation, and decapsulation. Cycle counts are averaged over 100 executions.

implementation	variant		Kyber-512 cc	Kyber-512 stack [B]	Kyber-768 cc	Kyber-768 stack [B]	Kyber-1024 cc	Kyber-1024 stack [B]
pqm4, [Alk+20]		K	458k	2 220	745k	3 100	1 188k	3 612
		E	553k	2 308	899k	2 780	1 373k	3 292
		D	513k	2 324	839k	2 804	1 294k	3 324
This work	speed	K	443k	4 272	718k	5 312	1 138k	6 336
		E	536k	5 376	870k	6 416	1 324k	7 432
		D	487k	5 384	796k	6 432	1 227k	7 448
	stack	K	444k	2 220	724k	2 736	1 149k	3 256
		E	540k	2 308	879k	2 808	1 341k	3 328
		D	492k	2 324	807k	2 824	1 246k	3 352

Dilithium: Scheme performance

Table: Cycle counts and stack usage for Dilithium. K, S, and V correspond to the key generation, signature generation, and signature verification. Cycle counts are averaged over 10000 executions.

implementation	variant		Dilithium2		Dilithium3		Dilithium5	
			cc	stack [B]	cc	stack [B]	cc	stack [B]
pqm4, [GKS20]		K	1 602k	38k	2 835k	61k	4 836k	98k
		S	4 336k	49k	6 721k	74k	9 037k	115k
		V	1 579k	36k	2 700k	58k	4 718k	93k
This work	speed	K	1 596k	8 508	2 827k	9 540	4 829k	11 696
		S	4 093k	49k	6 623k	69k	8 803k	116k
		V	1 572k	36k	2 692k	58k	4 707k	93k

Faster Kyber and Dilithium on the Cortex-M4

Amin Abdulrahman[1,2] Vincent Hwang[3,4] Matthias J. Kannwischer[3] Amber Sprenkels[5]

[1]Ruhr University Bochum, Germany

[2]Max Planck Institute for Security and Privacy, Bochum, Germany

[3]Academia Sinica, Taipei, Taiwan

[4]National Taiwan University, Taipei, Taiwan

[5]Digital Security Group, Radboud University, Nijmegen, The Netherlands

23rd June 2022

Bibliography I

[Abd+21] Amin Abdulrahman et al. *Multi-moduli NTTs for Saber on Cortex-M3 and Cortex-M4*. Cryptology ePrint Archive, Report 2021/995. https://ia.cr/2021/995. 2021.

[Alk+20] Erdem Alkim et al. 'Cortex-M4 Optimizations for {R,M}LWE Schemes'. In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2020.3 (June 2020), pp. 336–357. DOI: 10.13154/tches.v2020.i3.336-357. URL: https://tches.iacr.org/index.php/TCHES/article/view/8593.

[Bai+20] Shi Bai et al. *CRYSTALS-Dilithium: Algorithm Specifications And Supporting Documentation (version 3.0)*. Submission to round 3 of the NIST post-quantum project [Nat]. Oct. 2020.

[Bec+21] Hanno Becker et al. *Neon NTT: Faster Dilithium, Kyber, and Saber on Cortex-A72 and Apple M1*. Cryptology ePrint Archive, Report 2021/986. https://ia.cr/2021/986. 2021.

Bibliography II

[BKS19] Leon Botros, Matthias J. Kannwischer, and Peter Schwabe. 'Memory-Efficient High-Speed Implementation of Kyber on Cortex-M4'. In: *Progress in Cryptology – AFRICACRYPT 2019*. Ed. by Johannes Buchmann, Abderrahmane Nitaj, and Tajjeeddine Rachidi. Cham: Springer International Publishing, 2019, pp. 209–228. ISBN: 978-3-030-23696-0.

[Chu+21] Chi-Ming Marvin Chung et al. 'NTT Multiplication for NTT-unfriendly Rings: New Speed Records for Saber and NTRU on Cortex-M4 and AVX2'. In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2021.2 (Feb. 2021), pp. 159–188. DOI: 10.46586/tches.v2021.i2.159-188. URL: https://tches.iacr.org/index.php/TCHES/article/view/8791.

Bibliography III

[GKS20] Denisa O. C. Greconici, Matthias J. Kannwischer, and Amber Sprenkels. 'Compact Dilithium Implementations on Cortex-M3 and Cortex-M4'. In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2021.1 (Dec. 2020), pp. 1–24. DOI: 10.46586/tches.v2021.i1.1-24. URL: https://tches.iacr.org/index.php/TCHES/article/view/8725.

[Nat] National Institute of Standards and Technology. *Post-Quantum Cryptography Standardization Project*. Accessed: 2021-04-04. URL: https://csrc.nist.gov/projects/post-quantum-cryptography.

Backup