Parallel Small Polynomial Multiplication for Dilithium: A Faster Design and Implementation

Jieyu Zheng[1], Feng He[1], Shen Shiyu[1], Chenxi Xue[1], Yunlei Zhao[1]

[1]School of Computer Science, Fudan University

{jieyuzheng21,shenshiyu21,cxxue21}@m.fudan.edu.cn {fhe20,ylzhao}@fudan.edu.cn

Annual Computer Security Applications Conference(ACSAC)

December 5-9, 2022

Outline

- Introduction
- Motivation
- Small Polynomial Multiplication
- Experimental results
- Conclusion

Introduction

Dilithium

- One of third-round Signature finalists(The final Signature scheme to be standardized)
- Module-LWE and Module-SIS
- Small keys and signatures
- Operates in $\mathbb{R}_q = \mathbb{Z}_q[x]/(x^n + 1)$
    - Allows efficient polynomial multiplication with NTT
- Parameters: $n = 256, q = 8380417$

Motivation

Dilithium Sign and Verify

$\mathsf{Sign}(sk, M)$

$\mathbf{A} \in R_q^{k \times \ell} := \mathsf{ExpandA}(\rho)$ $\triangleright$ $\mathbf{A}$ is generated and stored in NTT Representation as $\hat{\mathbf{A}}$

$\mu \in \{0,1\}^{384} := \mathsf{CRH}(tr \parallel M)$

$\kappa := 0,\ (\mathbf{z}, \mathbf{h}) := \perp$

$\rho' \in \{0,1\}^{384} := \mathsf{CRH}(K \parallel \mu)$ (or $\rho' \leftarrow \{0,1\}^{384}$ for randomized signing)

while $(\mathbf{z}, \mathbf{h}) = \perp$ **do** $\triangleright$ Pre-compute $\hat{\mathbf{s}}_1 := \mathtt{NTT}(\mathbf{s}_1)$, $\hat{\mathbf{s}}_2 := \mathtt{NTT}(\mathbf{s}_2)$, and $\hat{\mathbf{t}}_0 := \mathtt{NTT}(\mathbf{t}_0)$

$\quad \mathbf{y} \in \tilde{S}_{\gamma_1}^{\ell} := \mathsf{ExpandMask}(\rho', \kappa)$

$\quad \mathbf{w} := \mathbf{A}\mathbf{y}$ $\triangleright$ $\mathbf{w} := \mathtt{NTT}^{-1}(\hat{\mathbf{A}} \cdot \mathtt{NTT}(\mathbf{y}))$

$\quad \mathbf{w}_1 := \mathsf{HighBits}_q(\mathbf{w}, 2\gamma_2)$

$\quad \tilde{c} \in \{0,1\}^{256} := \mathsf{H}(\mu \parallel \mathbf{w}_1)$

$\quad c \in B_\tau := \mathsf{SampleInBall}(\tilde{c})$ $\triangleright$ Store c in NTT representation as $\hat{c} = \mathtt{NTT}(c)$

$\quad \mathbf{z} := \mathbf{y} + c\mathbf{s}_1$ $\triangleright$ Compute $c\mathbf{s}_1$ as $\mathtt{NTT}^{-1}(\hat{c} \cdot \hat{\mathbf{s}}_1)$

$\quad \mathbf{r}_0 := \mathsf{LowBits}_q(\mathbf{w} - c\mathbf{s}_2, 2\gamma_2)$ $\triangleright$ Compute $c\mathbf{s}_2$ as $\mathtt{NTT}^{-1}(\hat{c} \cdot \hat{\mathbf{s}}_2)$

$\quad$ **if** $\|\mathbf{z}\|_\infty \geq \gamma_1 - \beta$ or $\|\mathbf{r}_0\|_\infty \geq \gamma_2 - \beta$, then $(\mathbf{z}, \mathbf{h}) := \perp$

$\quad$ **else**

$\quad\quad \mathbf{h} := \mathsf{MakeHint}_q(-c\mathbf{t}_0, \mathbf{w} - c\mathbf{s}_2 + c\mathbf{t}_0, 2\gamma_2)$ $\triangleright$ Compute $c\mathbf{t}_0$ as $\mathtt{NTT}^{-1}(\hat{c} \cdot \hat{\mathbf{t}}_0)$

$\quad\quad$ **if** $\|c\mathbf{t}_0\|_\infty \geq \gamma_2$ **or** the # of 1's in $\mathbf{h}$ is greater than ω, then $(\mathbf{z}, \mathbf{h}) := \perp$

$\quad \kappa := \kappa + \ell$

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

$\mathsf{Verify}(pk, M, \sigma = (\mathbf{z}, \mathbf{h}, \tilde{c}))$

$\mathbf{A} \in R_q^{k \times \ell} := \mathsf{ExpandA}(\rho)$ $\triangleright$ $\mathbf{A}$ is generated and stored in NTT Representation as $\hat{\mathbf{A}}$

$\mu \in \{0,1\}^{384} := \mathsf{CRH}(\mathsf{CRH}(\rho \parallel \mathbf{t}_1) \parallel M)$

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

$\mathbf{w}_1' := \mathsf{UseHint}_q(\mathbf{h}, \mathbf{Az} - c\mathbf{t}_1 \cdot 2^d, 2\gamma_2)$ $\triangleright$ Compute as $\mathtt{NTT}^{-1}(\hat{\mathbf{A}} \cdot \mathtt{NTT}(\mathbf{z}) - \mathtt{NTT}(c) \cdot \mathtt{NTT}(\mathbf{t}_1 \cdot 2^d))$

return $[\![\|\mathbf{z}\|_\infty < \gamma_1 - \beta]\!]$ **and** $[\![\tilde{c} = \mathsf{H}(\mu \parallel \mathbf{w}_1')]\!]$ **and** $[\![\#$ of 1's in $\mathbf{h}$ is $\leq \omega]\!]$

Small Polynomial Multiplication

Small Polynomial

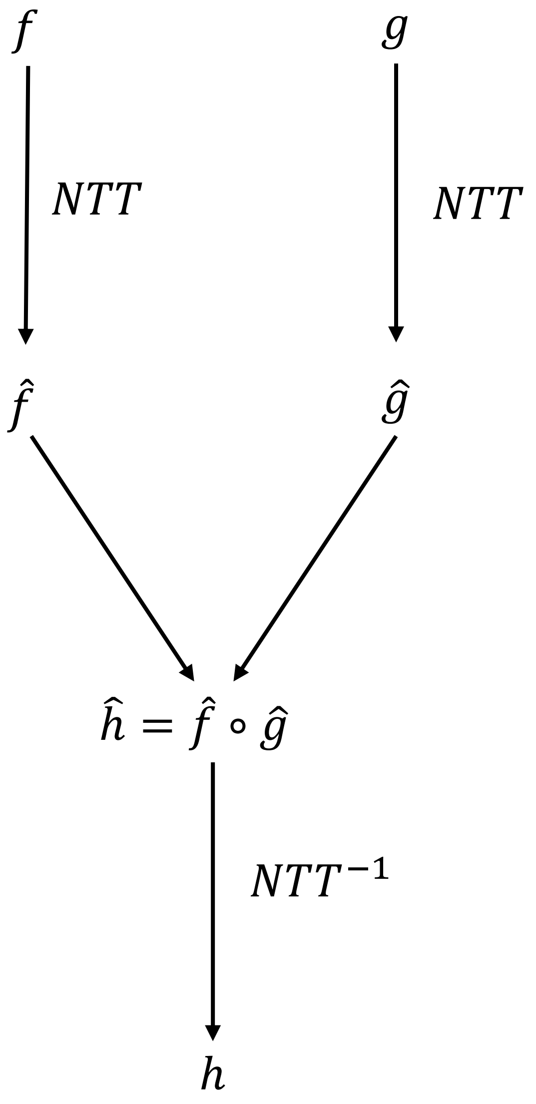
- Coefficients are much smaller than q.
- Most coefficients are 0, few are ± 1.

$\mathsf{SampleInBall}(\rho)$

```
01 Initialize c = c_0 c_1 ... c_255 = 00...0
02 for i := 256 − τ to 255
03     j ← {0, 1, ..., i}
04     s ← {0, 1}
05     c_i := c_j
06     c_j := (−1)^s
07 return c
```

Motivation

NTT Technique

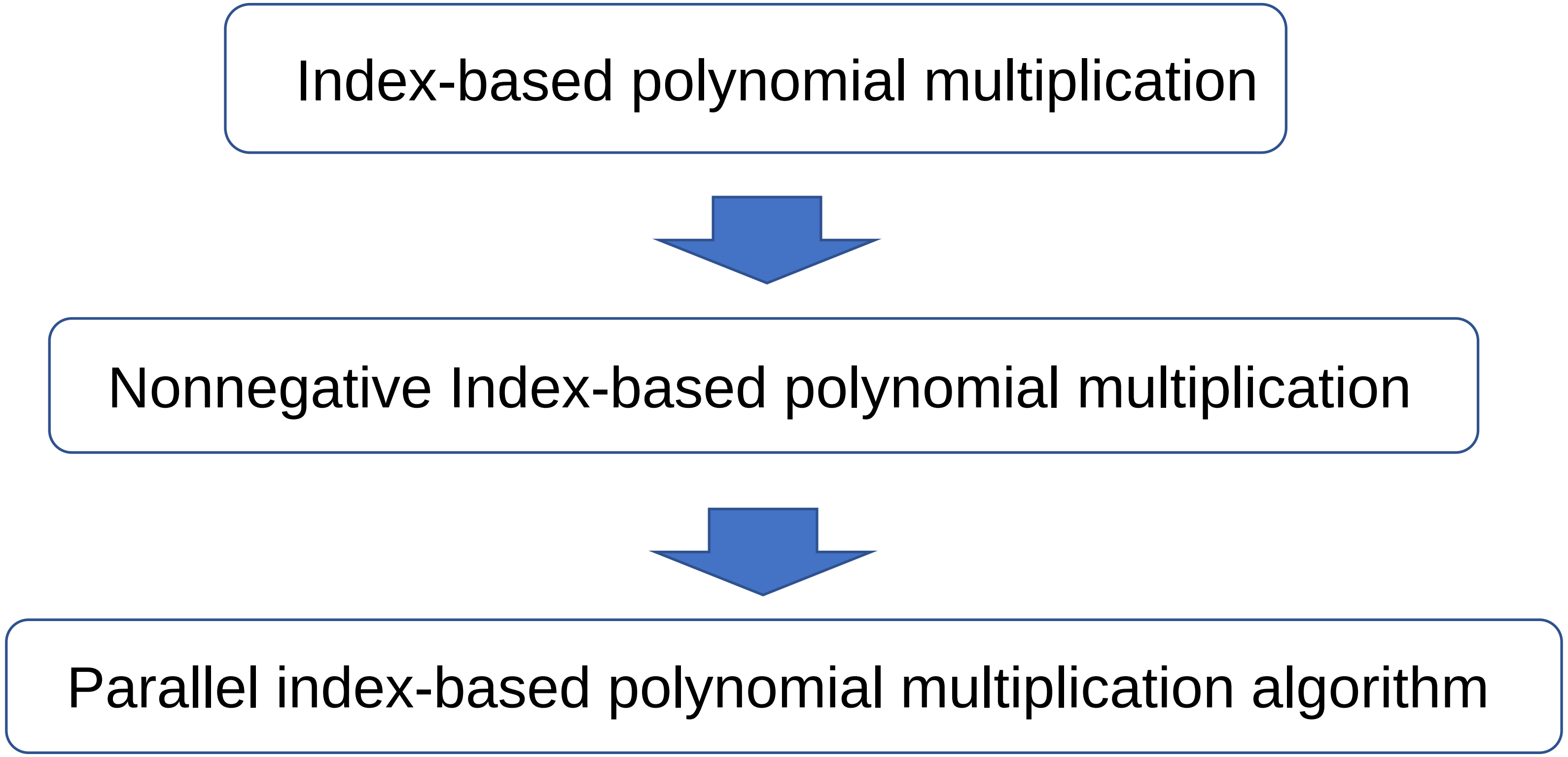


The previous technique to speed up polynomial multiplication is Number Theoretic Transform(NTT).

Can we derive a polynomial multiplication for small polynomial?

The answer is “**Yes!**”

Small Polynomial Multiplication: Technique Overview



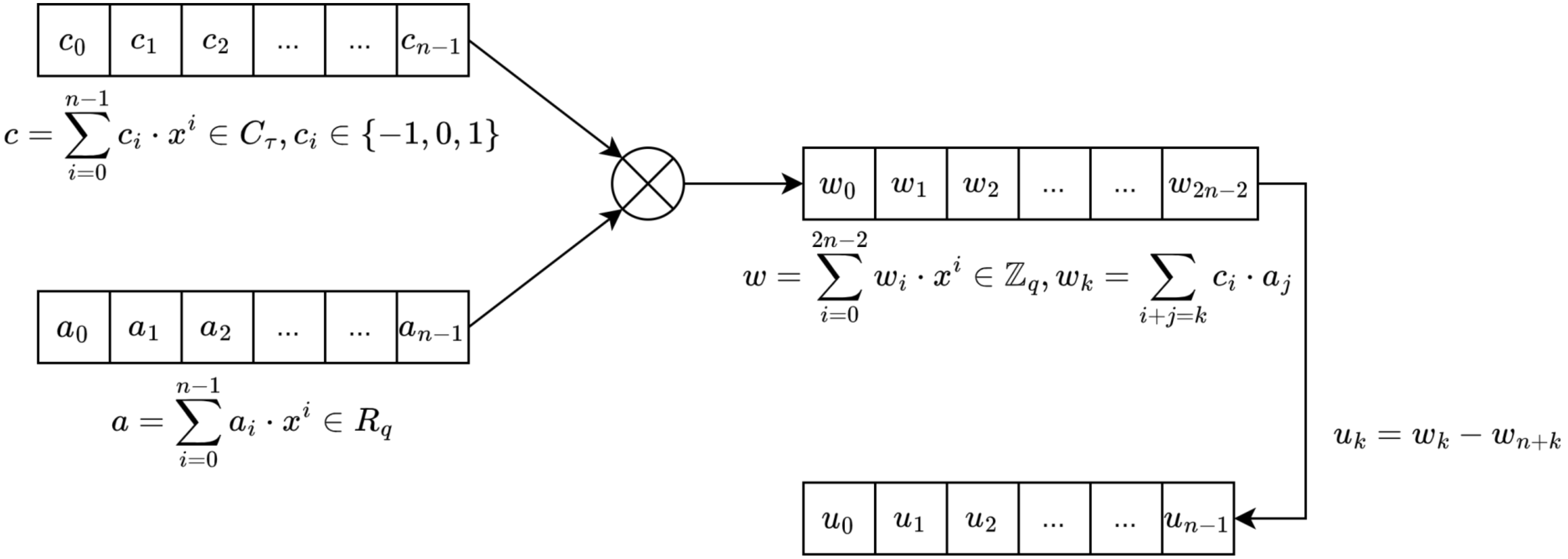
Index-based Small Polynomial Multiplication

- For $u = \sum_{i=0}^{n-1} u \cdot x^i$, $0 \leq i \leq n-2$ we have :

$$u_i = \sum_{j=0}^{i} c_j \cdot a_{i-j} - \sum_{j=i+1}^{n-1} c_j \cdot a_{n+i-j} = \sum_{j=0}^{i} c_j \cdot a_{i-j} + \sum_{j=i+1}^{n-1} c_j \cdot (-a_{n+i-j})$$

$c_j \cdot a_{i-j}$ and $c_j \cdot a_{n+i-j}$ can be replaced by a_{i-j} and a_{n+i-j} ($c_j = 1$), $-a_{i-j}$ and $-a_{n+i-j}$ ($c_j = -1$).

- For $i = n-1$, we have $u_i = \sum_{j=0}^{n-1} c_j \cdot a_{i-j}$.



Index-based Small Polynomial Multiplication

Algorithm 6 An alternative index-based polynomial multiplication algorithm for computing **ca**

Input: $\mathbf{c} = \sum_{i=0}^{n-1} c_i \cdot x^i \in B_\tau$, $\mathbf{a} = \sum_{i=0}^{n-1} a_i \cdot x^i \in \mathcal{R}_q$

Output: $\mathbf{u} = \mathbf{c} \cdot \mathbf{a} \in \mathcal{R}_q$

for $i \in \{0, 1, \cdots, n-1\}$ **do**
  $w_i := 0$
  $v_i := a_i$
  $v_{i-n} := -a_i$
end for
for $i \in \{0, 1, \cdots, n-1\}$ **do**
  if $c_i = 1$ **then**
    for $j \in \{0, 1, \cdots, n-1\}$ **do**
      $w_j := w_j + v_{j-i}$
    end for
  end if
  if $c_i = -1$ **then**
    for $j \in \{0, 1, \cdots, n-1\}$ **do**
      $w_j := w_j - v_{j-i}$
    end for
  end if
end for
for $i \in \{0, 1, \cdots, n-1\}$ **do**
  $u_i := w_i \pmod q$ ▷ $u_i \in \mathbb{F}_q$
end for
$\mathbf{u} := \sum_{i=0}^{n-1} u_i \cdot x^i$ ▷ $\mathbf{u} \in \mathcal{R}_q$
return u

$$u_i = \sum_{j=0}^{i} c_j \cdot a_{i-j} - \sum_{j=i+1}^{n-1} c_j \cdot a_{n+i-j} = \sum_{j=0}^{i} c_j \cdot a_{i-j} + \sum_{j=i+1}^{n-1} c_j \cdot (-a_{n+i-j})$$

$$v_i = \begin{cases} a_i, & 0 \le i \le n-1 \\ -a_{n+i}, & 1-n \le i \le -1 \end{cases}$$

$$u_i = \sum_{j=0}^{n-1} c_j \cdot v_{i-j}$$

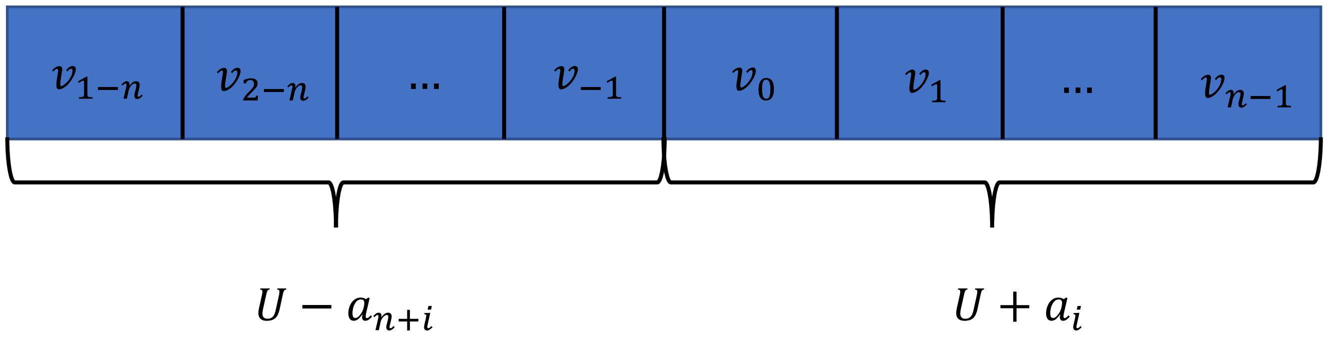
$$u_i = \sum_{c_j=1} v_{i-j} + \sum_{c_j=-1} (-v_{i-j})$$

Addition and subtraction replace multiplication

Nonnegative Small Polynomial Multiplication

- The above algorithm is not suitable for deriving parallel algorithm.

- Make algorithm nonnegative.



- U is upper bound of coefficients.

Algorithm 7 An index-based polynomial multiplication algorithm with translations

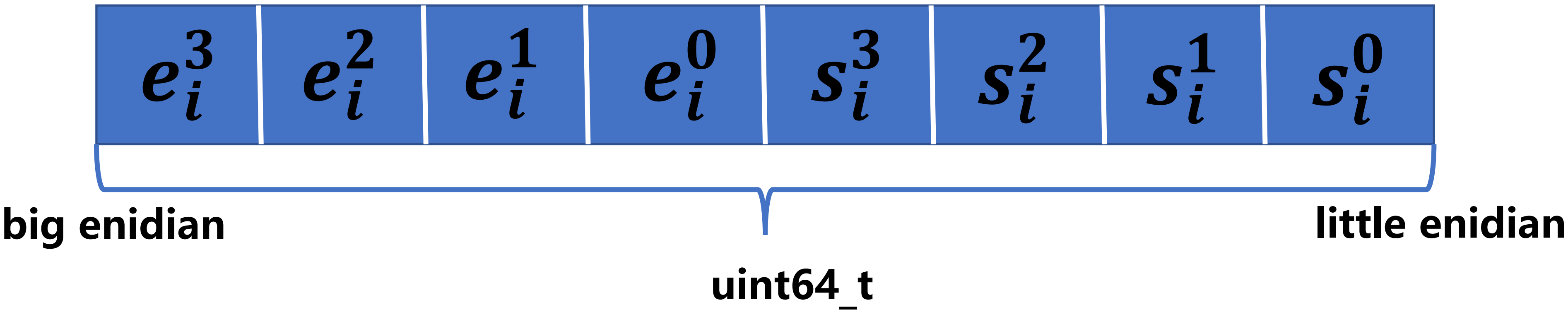
Input: $\mathbf{c} = \sum_{i=0}^{n-1} c_i \cdot x^i \in B_\tau$, $\mathbf{a} = \sum_{i=0}^{n-1} a_i \cdot x^i \in \mathcal{R}_q$

Output: $\mathbf{u} = \mathbf{c} \cdot \mathbf{a} \in \mathcal{R}_q$

for $i \in \{0, 1, \cdots, n-1\}$ **do**
  $w_i := 0$
  $v_i := U + a_i$
  $v_{i-n} := U - a_i$
end for
for $i \in \{0, 1, \cdots, n-1\}$ **do**
  if $c_i = 1$ **then**
    for $j \in \{0, 1, \cdots, n-1\}$ **do**
      $w_j := w_j + v_{j-i}$
    end for
  end if
  if $c_i = -1$ **then**
    for $j \in \{0, 1, \cdots, n-1\}$ **do**
      $w_j := w_j + (2U - v_{j-i})$
    end for
  end if
end for
for $i \in \{0, 1, \cdots, n-1\}$ **do**
  $u_i := w_i - \tau U \pmod{q}$ ▷ $u_i \in \mathbb{F}_q$
end for
$\mathbf{u} := \sum_{i=0}^{n-1} u_i \cdot x^i$ ▷ $\mathbf{u} \in \mathcal{R}_q$
return u

Parallel Small Polynomial Multiplication

- Compute $c \cdot \vec{\boldsymbol{a}}$
  - $c \in B_\tau, c_i \in \{-1,0,1\}$
  - $\vec{\boldsymbol{a}} = [a^{(0)}, \dots, a^{(r-1)}]^T \in R_q^r$ is a polynomial vector, there exists an constant U that $\left\|\boldsymbol{a}^{(j)}\right\|_\infty \le U, \ \ \forall j \in \{0,1,\dots,r-1\}$, r is the number of polynomial that a word (64bit) can pack.



Parallel Small Polynomial Multiplication

Algorithm 9 A parallel index-based polynomial multiplication algorithm with translations

Input: $\left(\mathbf{c}, \vec{\mathbf{a}}\right)$, where

- $\mathbf{c} = \sum_{i=0}^{n-1} c_i \cdot x^i \in B_\tau$;
- $\vec{\mathbf{a}} = \left\{\mathbf{a}^{(j)}\right\} \in \mathcal{R}_q^r$;
- Every $\mathbf{a}^{(j)} = \sum_{i=0}^{n-1} a_i^{(j)} \cdot x^i \in \mathcal{R}_q$;
- Every $a_i^{(j)} \in \{-U, \cdots, U\}$

Output: $\vec{\mathbf{u}} = \left[\mathbf{u}^{(0)}, \cdots, \mathbf{u}^{(r-1)}\right]^T \in \mathcal{R}_q^r$, where

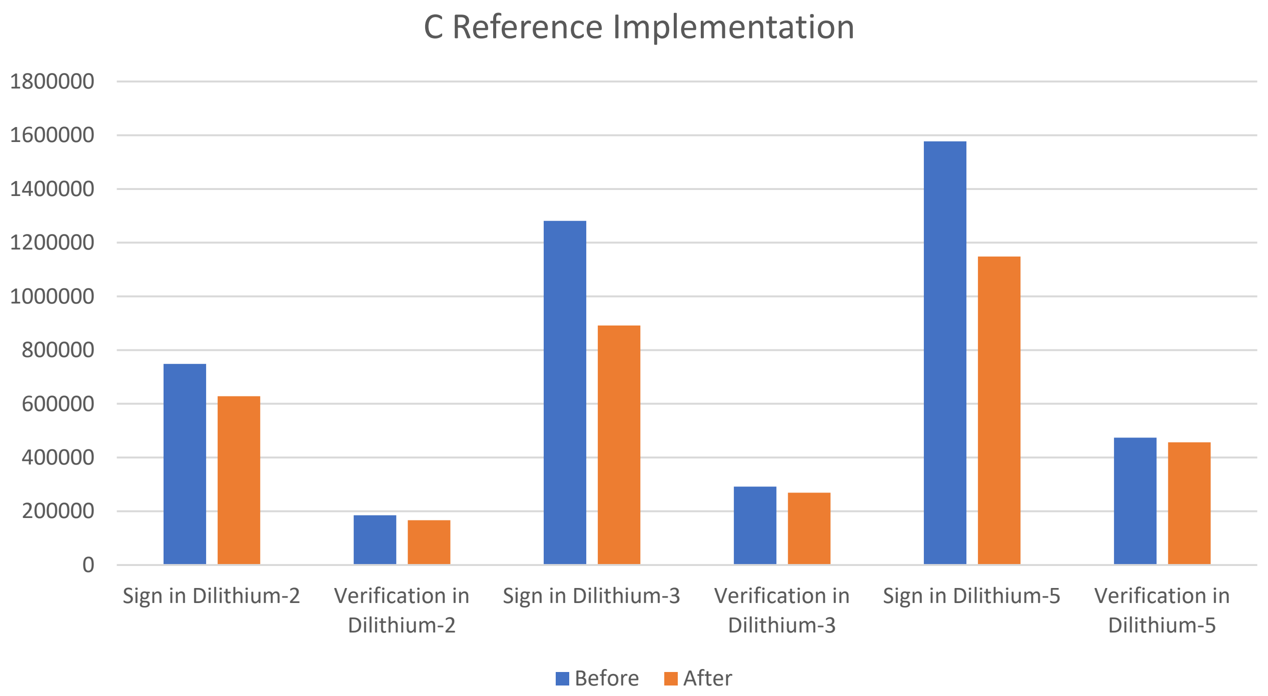
- $\mathbf{u}^{(j)} = \mathbf{c} \cdot \mathbf{a}^{(j)} \in \mathcal{R}_q$;

for $i \in \{0, 1, \cdots, n-1\}$ **do**
    $w_i := 0$
    $v_i := 0$
    $v_{i-n} := 0$
    for $j = 0$ **to** $r-1$ **do**
        $v_i := v_i \cdot M + \left(U + a_i^{(j)}\right)$
        $v_{i-n} := v_{i-n} \cdot M + \left(U - a_i^{(j)}\right)$
    end for
end for
$\gamma := 2U \cdot \frac{M^r-1}{M-1}$ ▷ $\gamma \in \mathbb{Z}^{>0}$

Pack vector coefficients

for $i \in \{0, 1, \cdots, n-1\}$ **do**
    if $c_i = 1$ **then**
        for $j \in \{0, 1, \cdots, n-1\}$ **do**
            $w_j := w_j + v_{j-i}$
        end for
    end if
    if $c_i = -1$ **then**
        for $j \in \{0, 1, \cdots, n-1\}$ **do**
            $w_j := w_j + (\gamma - v_{j-i})$ ▷ $\gamma = 2U \cdot \frac{M^r-1}{M-1}$
        end for
    end if
end for
for $i \in \{0, 1, \cdots, n-1\}$ **do**
    $t := w_i$
    for $j = 0$ **to** $r-1$ **do**
        $u_i^{(r-1-j)} := (t \bmod M) - \tau U \pmod{q}$
        $t := \lfloor t/M \rfloor$
    end for
end for
for $j \in \{0, 1, \cdots, r-1\}$ **do**
    $\mathbf{u}^{(j)} := \sum_{i=0}^{n-1} u_i^{(j)} \cdot x^i$
end for
$\vec{\mathbf{u}} := \left[\mathbf{u}^{(0)}, \cdots, \mathbf{u}^{(r-1)}\right]^T$
return $\vec{\mathbf{u}}$

Experimental Results

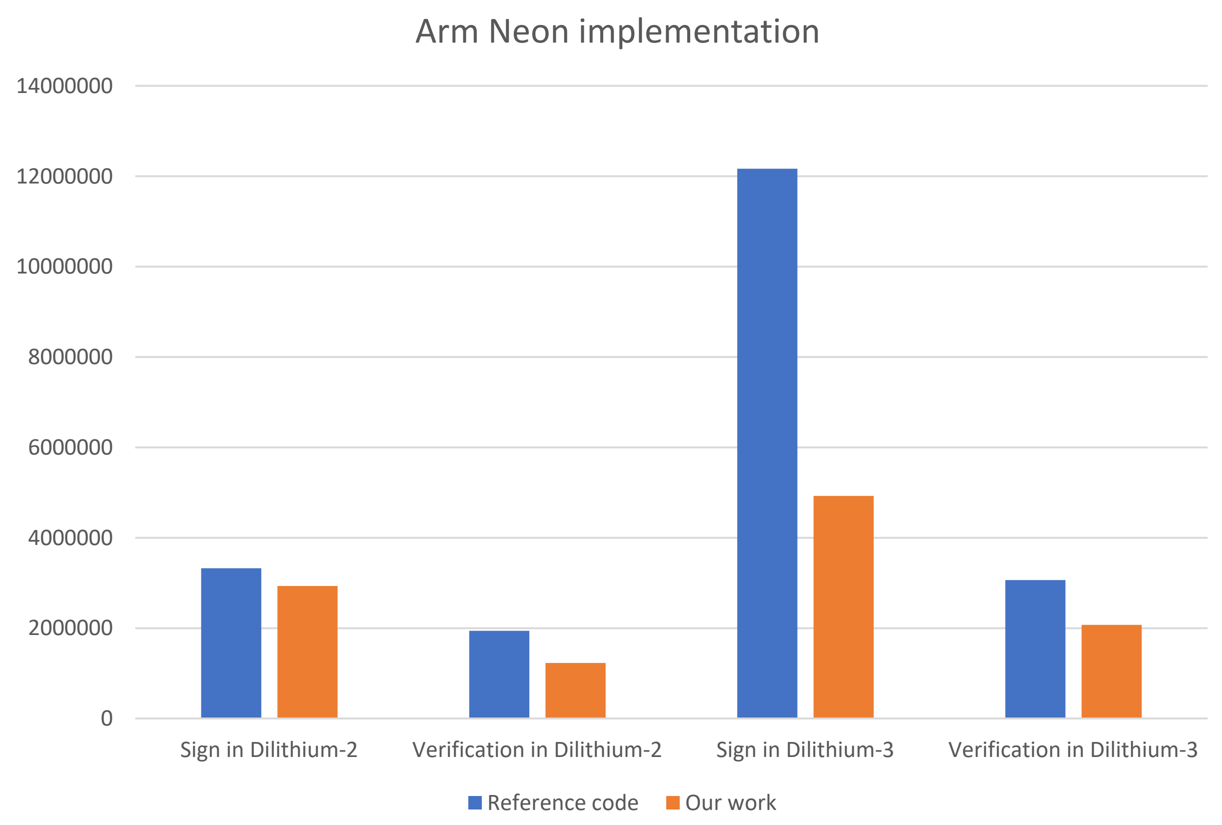


	Before	After
Sign in Dilithium-2	748500	628171
Verification in Dilithium-2	185022	166419
Sign in Dilithium-3	1281313	891613
Verification in Dilithium-3	291921	269118
Sign in Dilithium-5	1577046	1148236
Verification in Dilithium-5	474205	456961

Table 5: Reference Implementation Comparitive Results in cpucycles.

- For Dilithium-2, we achieve **18%** speed-up in Sign, **19%** in Verify.
- For Dilithium-3, we achieve **30%** speed-up in Sign, **7%** in Verify.
- For Dilithium-5, we achieve **27%** speed-up in Sign, **3%** speed-up in Verify.

Experimental Results

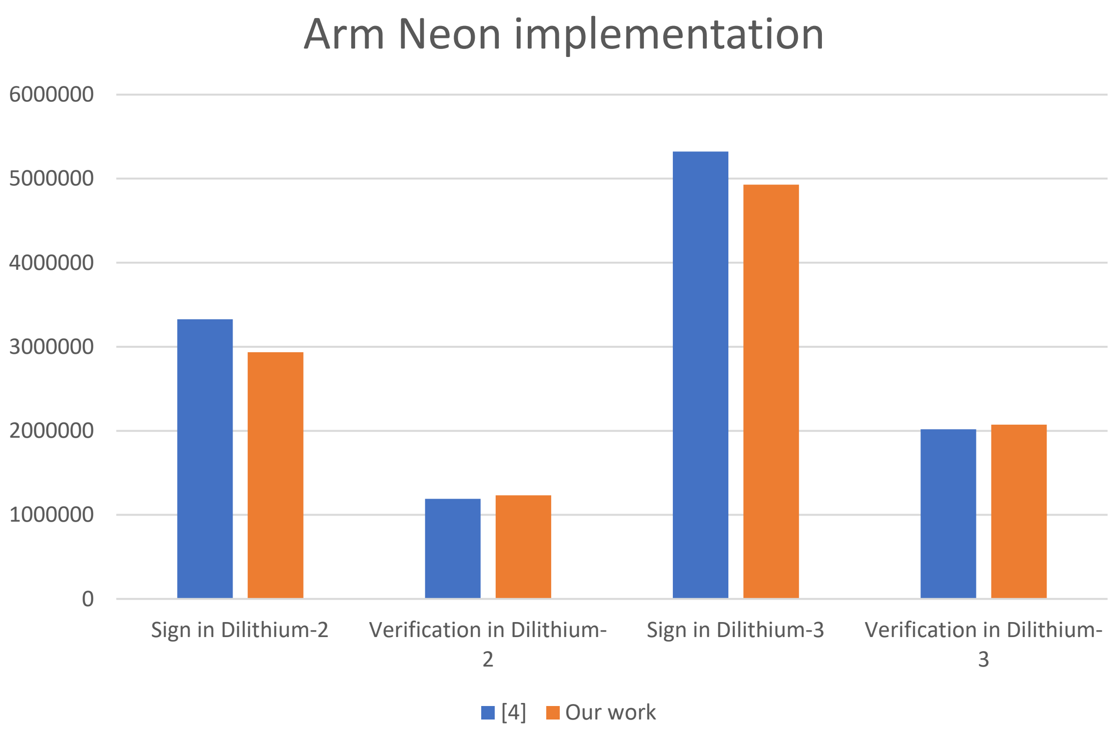


	Reference code	Our work
Sign in Dilithium-2	8223359	2934124
Verification in Dilithium-2	1941673	1231796
Sign in Dilithium-3	12166847	4927678
Verification in Dilithium-3	3063575	2073518

Table 6: Our neon implementation and reference Implementation Comparitive Results in cpucycles.

- For Dilithium-2, we achieve **64%** speed-up in Sign, **50%** in Verify.
- For Dilithium-3, we achieve **60%** speed-up in Sign, **32%** in Verify.

Experimental Results



	[4]	Our work
Sign in Dilithium-2	3327206	2934124
Verification in Dilithium-2	1191080	1231796
Sign in Dilithium-3	5321618	4927678
Verification in Dilithium-3	2018945	2073518

Table 7: Arm neon Implementation Comparitive Results in cpucycles.

- Compared with the state-of-art implementation.
- For Dilithium-2, we achieve **13.4%** speed-up in Sign.
- For Dilithium-3, we achieve **8%** speed-up in Sign.

[4] Hanno Becker, Vincent Hwang, Matthias J. Kannwischer, Bo-Yin Yang, and Shang Yi Yang. 2022. Neon NTT: Faster Dilithium, Kyber, and Saber on Cortex-A72 and Apple M1. IACR Trans. Cryptogr. Hardw. Embed. Syst. 2022, 1 (2022), 221–244.https://doi.org/10.46586/tches.v2022.i1.221-244

Experimental Results

- **Polynomial vector multiplication in Dilithium-3**

	Our Algorithm	NTT
$\mathbf{c}\vec{\mathbf{s}}$&$\mathbf{c}\vec{\mathbf{e}}$ in Dilithium-3	8477	73924
$\mathbf{c}\vec{\mathbf{t}}_0$ in Dilithium-3	14132	88198
$\mathbf{c}\vec{\mathbf{t}}_1$ in Dilithium-3	11760	90992

- Compared with the NTT technique implementation.
- For $c\vec{s}$&$c\vec{e}$, we achieve **88%** speed-up.
- For $c\vec{t_0}$, we achieve **84%** speed-up.
- For $c\vec{t_1}$, we achieve **87%** speed-up.

Conclusion

- We exhibit a small polynomial multiplication parallel algorithm.
- We complete the C reference implementation.
- We improve the algorithm by Neon vector extension on the Cortex-A72 platform.
- Our Arm Neon implementation of Dilithium achieves a new record of fast Dilithium implementation.

Parallel Small Polynomial Multiplication for Dilithium: A Faster Design and Implementation

THANKS!

Questions?