Multi-moduli NTTs for Saber on Cortex-M3 and Cortex-M4

Amin Abdulrahman Jiun-Peng Chen Yu-Jia Chen Vincent Hwang
Matthias J. Kannwischer Bo-Yin Yang

CHES 2022, Leuven, Belgium

Organization of This Talk

Introduction

Time–Memory Tradeoffs

- Polynomial Multiplications on Cortex-M4
- `MatrixVectorMul`

First–Order Masked `MatrixVectorMul` and `InnerProd`

Saber on Cortex-M3

Results

Introduction

Saber

- $R_q = \mathbb{Z}_{8192}[x]/\langle x^{256}+1\rangle$
- Parameters (l, μ) varies from security levels (other parameters omitted in this talk).
  - `LightSaber` : $(l, \mu) = (2, 10)$
  - `Saber` : $(l, \mu) = (3, 8)$
  - `FireSaber` : $(l, \mu) = (4, 6)$
- $A \in R_q^{l \times l}, s, s' \in R_q^l$.
  - Key generation: $A^T s$
  - Encryption: As'

NTT–Based `MatrixVectorMul` for Saber

- Find an NTT-friendly modulus q' such that $A^T s$ in $\mathbb{Z}$ is the same as in $\mathbb{Z}_{q'}$
  - NTT–firendly: next slide
  - Signed arithmetic: choose $q' > 2 \cdot \frac{8192}{2} \cdot \frac{\mu}{2} \cdot l$
- Compute $A^T s = \mathtt{NTT}^{-1}\left(\mathtt{NTT}(A^T) \cdot \mathtt{NTT}(s)\right)$
  - $l^2 + l$ $\mathtt{NTT}$s
  - l $\mathtt{NTT}^{-1}$s
  - l^2 base multiplications

Number–Theoretic Transforms i

- Ring R, invertible $\zeta \in R$
- $n \perp \mathrm{char}(R)$, principal n-th root of unity ω_n ($\forall 1 \leq i < n, \sum_{j=0}^{n-1} \omega_n^{ij} = 0$). Equivalently, for $R = \mathbb{Z}_q$ with prime factorization $q = \prod_{i=0}^{l-1} p_i^{d_i}$, $n | \mathbf{0}(q) := \gcd(p_i - 1)_{0 \leq i < l}$ [AB74].
- $R[x]/\langle x^n - \zeta^n \rangle \cong \prod_{i=0}^{n-1} R[x]/\langle x - \zeta\omega_n^i \rangle : \boldsymbol{a}(x) \mapsto \boldsymbol{a}(\zeta\omega_n^i)_i$
- Cooley–Tukey FFT:

$$
\begin{aligned}
R[x]/\langle x^{n_0 n_1} - \zeta^{n_0 n_1} \rangle &\cong \prod_{i_0=0}^{n_0-1} R[x]/\langle x^{n_1} - \zeta^{n_1}\omega_n^{i_0 n_1} \rangle \\
&\cong \prod_{i_0=0}^{n_0-1} \prod_{i_1=0}^{n_1-1} R[x]/\langle x - \zeta\omega_n^{i_0 + i_1 n_0} \rangle
\end{aligned}
$$

Number–Theoretic Transforms ii

$$R[x]\Big/\left\langle x^{2^k}-\zeta^{2^k}\right\rangle \cong \prod_{i_0=0}^{1} R[x]\Big/\left\langle x^{2^{k-1}}-\zeta^{2^{k-1}}\omega_2^{i_0}\right\rangle$$

$$\cong \prod_{i_0,i_1=0}^{1} R[x]\Big/\left\langle x^{2^{k-2}}-\zeta^{2^{k-2}}\omega_4^{i_0+2i_1}\right\rangle$$

$$\cong \prod_{i_0,\ldots,i_{k-1}=0}^{1} R[x]\Big/\left\langle x-\zeta\omega_{2^k}^{\sum_{j=0}^{k-1}2^j i_j}\right\rangle$$

- k isomorphisms of product rings
- Each isomorphism takes $O(2^k)$ time $\Longrightarrow O(k2^k)$ time (or $O(n\lg n)$ where $n=2^k$)

Number–Theoretic Transforms iii

- $R = \mathbb{Z}_{q_0 q_1}$, $R_0 = \mathbb{Z}_{q_0}$, $R_1 = \mathbb{Z}_{q_1}$, $q_0 \perp q_1$
- $(\zeta_0, \zeta_1) = (\zeta \bmod q_0, \zeta \bmod q_1), (\omega_{0:n}, \omega_{1:n}) = (\omega_n \bmod q_0, \omega_n \bmod q_1)$
- $\texttt{NTT} := \boldsymbol{a}(x) \mapsto \boldsymbol{a}(\zeta\omega_n^i)_i$, $\texttt{NTT}_0 := \boldsymbol{a}(x) \mapsto \boldsymbol{a}(\zeta_0\omega_{0:n}^{i_0})_{i_0}$, $\texttt{NTT}_1 := \boldsymbol{a}(x) \mapsto \boldsymbol{a}(\zeta_1\omega_{1:n}^{i_1})_{i_1}$

$$\frac{R[x]}{\langle x^n - \zeta^n \rangle} \xrightarrow{\texttt{NTT}} \prod_{i=0}^{n-1} \frac{R[x]}{\langle x - \zeta\omega_n^i \rangle}$$

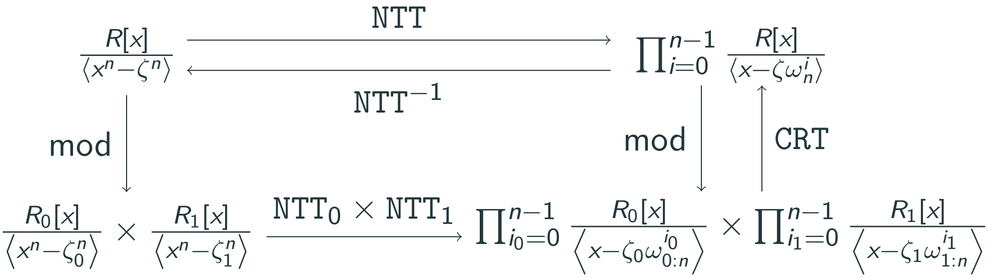
$$\frac{R[x]}{\langle x^n - \zeta^n \rangle} \xrightarrow{\text{mod}} \frac{R_0[x]}{\langle x^n - \zeta_0^n \rangle} \times \frac{R_1[x]}{\langle x^n - \zeta_1^n \rangle}$$

$$\frac{R_0[x]}{\langle x^n - \zeta_0^n \rangle} \times \frac{R_1[x]}{\langle x^n - \zeta_1^n \rangle} \xrightarrow{\texttt{NTT}_0 \times \texttt{NTT}_1} \prod_{i_0=0}^{n-1} \frac{R_0[x]}{\left\langle x - \zeta_0\omega_{0:n}^{i_0} \right\rangle} \times \prod_{i_1=0}^{n-1} \frac{R_1[x]}{\left\langle x - \zeta_1\omega_{1:n}^{i_1} \right\rangle}$$

$$\prod_{i_0=0}^{n-1} \frac{R_0[x]}{\left\langle x - \zeta_0\omega_{0:n}^{i_0} \right\rangle} \times \prod_{i_1=0}^{n-1} \frac{R_1[x]}{\left\langle x - \zeta_1\omega_{1:n}^{i_1} \right\rangle} \xrightarrow{\texttt{CRT}} \prod_{i=0}^{n-1} \frac{R[x]}{\langle x - \zeta\omega_n^i \rangle}$$

Number–Theoretic Transforms iv

What we will do next.



Time–Memory Tradeoffs

Memory for Polynomials

Assumptions:

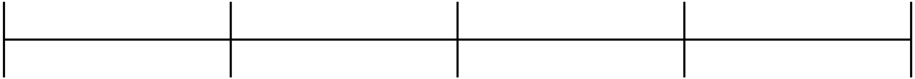
- Secrete polynomials are stored in their 4-bit form.
- Public polynomials are store in their 16-bit form.
- Public polynomials are only used once. Memory can be re-used.
- Expand to 32-bit when needed.

Stack usage:

- Memory for buffers.
- Memory for public polynomials.
- We ignore the memory for secrete polynomials.

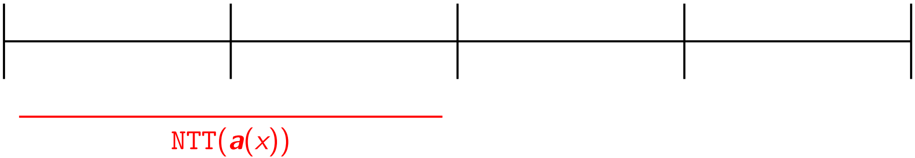
Stack Usage of 32-bit NTT-Based Polynomial Multiplications

- Each line segment = 4096 bits, 16384 bits in total.
- A size-256 poly. of 32-bit coeffs. is stored in two segments.
- Compute $\boldsymbol{a}(x)\boldsymbol{b}(x) = \mathtt{NTT}^{-1}(\mathtt{NTT}(\boldsymbol{a}(x)) \cdot \mathtt{NTT}(\boldsymbol{b}(x)))$
- Expand $\boldsymbol{a}(x), \boldsymbol{b}(x)$ to 32-bit first



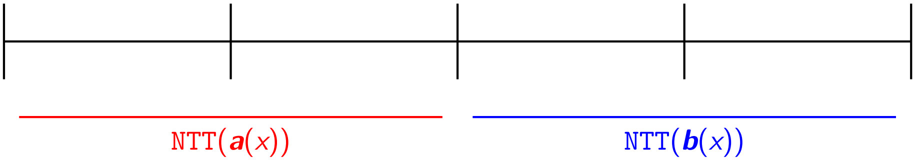
Stack Usage of 32-bit NTT-Based Polynomial Multiplications

- Each line segment = 4096 bits, 16384 bits in total.
- A size-256 poly. of 32-bit coeffs. is stored in two segments.
- Compute $\boldsymbol{a}(x)\boldsymbol{b}(x) = \texttt{NTT}^{-1}(\texttt{NTT}(\boldsymbol{a}(x)) \cdot \texttt{NTT}(\boldsymbol{b}(x)))$
- Expand $\boldsymbol{a}(x)$, $\boldsymbol{b}(x)$ to 32-bit first



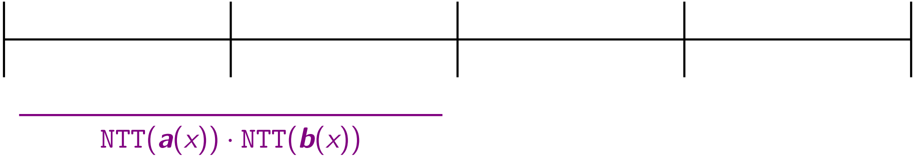
Stack Usage of 32-bit NTT-Based Polynomial Multiplications

- Each line segment = 4096 bits, 16384 bits in total.
- A size-256 poly. of 32-bit coeffs. is stored in two segments.
- Compute $\boldsymbol{a}(x)\boldsymbol{b}(x) = \texttt{NTT}^{-1}(\texttt{NTT}(\boldsymbol{a}(x)) \cdot \texttt{NTT}(\boldsymbol{b}(x)))$
- Expand $\boldsymbol{a}(x)$, $\boldsymbol{b}(x)$ to 32-bit first



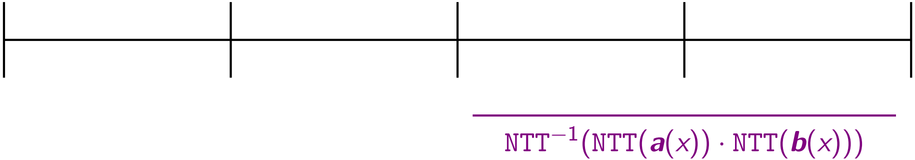
Stack Usage of 32-bit NTT-Based Polynomial Multiplications

- Each line segment = 4096 bits, 16384 bits in total.
- A size-256 poly. of 32-bit coeffs. is stored in two segments.
- Compute $\boldsymbol{a}(x)\boldsymbol{b}(x) = \mathtt{NTT}^{-1}(\mathtt{NTT}(\boldsymbol{a}(x)) \cdot \mathtt{NTT}(\boldsymbol{b}(x)))$
- Expand $\boldsymbol{a}(x)$, $\boldsymbol{b}(x)$ to 32-bit first



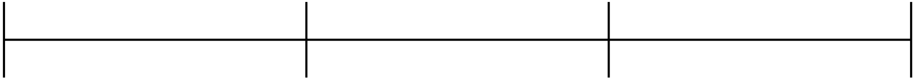
Stack Usage of 32-bit NTT-Based Polynomial Multiplications

- Each line segment = 4096 bits, 16384 bits in total.
- A size-256 poly. of 32-bit coeffs. is stored in two segments.
- Compute $\boldsymbol{a}(x)\boldsymbol{b}(x) = \mathtt{NTT}^{-1}(\mathtt{NTT}(\boldsymbol{a}(x)) \cdot \mathtt{NTT}(\boldsymbol{b}(x)))$
- Expand $\boldsymbol{a}(x), \boldsymbol{b}(x)$ to 32-bit first



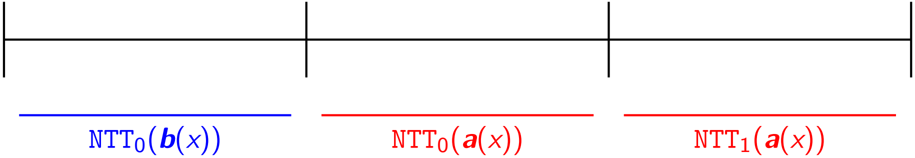
Stack Usage of 16-bit NTT-Based Polynomial Multiplications

- Each line segment = 4096 bits, 12288 bits in total.
- A size-256 poly. of 16-bit coeffs. is stored in a segment.
- Compute
  $\boldsymbol{a}(x)\boldsymbol{b}(x) = \mathrm{CRT}\left(\mathtt{NTT}_0^{-1}(\mathtt{NTT}_0(\boldsymbol{a}(x)) \cdot \mathtt{NTT}_0(\boldsymbol{b}(x))), \mathtt{NTT}_1^{-1}(\mathtt{NTT}_1(\boldsymbol{a}(x)) \cdot \mathtt{NTT}_1(\boldsymbol{b}(x)))\right)$
- Notice $(\boldsymbol{a}(x)\boldsymbol{b}(x) \bmod q_0, \boldsymbol{a}(x)\boldsymbol{b}(x) \bmod q_1) =$
  $(\mathtt{NTT}_0^{-1}(\mathtt{NTT}_0(\boldsymbol{a}(x)) \cdot \mathtt{NTT}_0(\boldsymbol{b}(x))), \mathtt{NTT}_1^{-1}(\mathtt{NTT}_1(\boldsymbol{a}(x)) \cdot \mathtt{NTT}_1(\boldsymbol{b}(x))))$



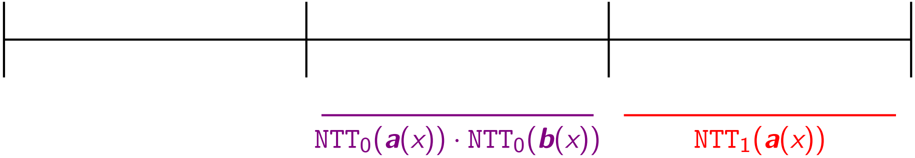
Stack Usage of 16-bit NTT-Based Polynomial Multiplications

- Each line segment = 4096 bits, 12288 bits in total.
- A size-256 poly. of 16-bit coeffs. is stored in a segment.
- Compute
  $\boldsymbol{a}(x)\boldsymbol{b}(x) = \mathrm{CRT}\left(\mathtt{NTT}_0^{-1}(\mathtt{NTT}_0(\boldsymbol{a}(x)) \cdot \mathtt{NTT}_0(\boldsymbol{b}(x))), \mathtt{NTT}_1^{-1}(\mathtt{NTT}_1(\boldsymbol{a}(x)) \cdot \mathtt{NTT}_1(\boldsymbol{b}(x)))\right)$
- Notice $(\boldsymbol{a}(x)\boldsymbol{b}(x) \bmod q_0, \boldsymbol{a}(x)\boldsymbol{b}(x) \bmod q_1) =$
  $(\mathtt{NTT}_0^{-1}(\mathtt{NTT}_0(\boldsymbol{a}(x)) \cdot \mathtt{NTT}_0(\boldsymbol{b}(x))), \mathtt{NTT}_1^{-1}(\mathtt{NTT}_1(\boldsymbol{a}(x)) \cdot \mathtt{NTT}_1(\boldsymbol{b}(x))))$



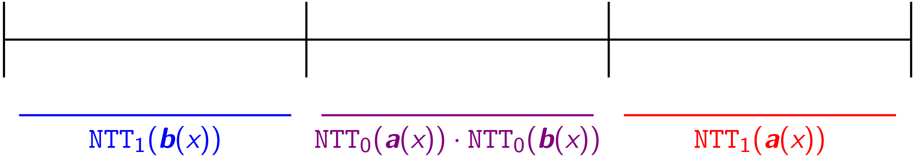
Stack Usage of 16-bit NTT-Based Polynomial Multiplications

- Each line segment = 4096 bits, 12288 bits in total.
- A size-256 poly. of 16-bit coeffs. is stored in a segment.
- Compute
  $\boldsymbol{a}(x)\boldsymbol{b}(x) = \mathrm{CRT}\left(\mathtt{NTT}_0^{-1}(\mathtt{NTT}_0(\boldsymbol{a}(x)) \cdot \mathtt{NTT}_0(\boldsymbol{b}(x))), \mathtt{NTT}_1^{-1}(\mathtt{NTT}_1(\boldsymbol{a}(x)) \cdot \mathtt{NTT}_1(\boldsymbol{b}(x)))\right)$
- Notice $(\boldsymbol{a}(x)\boldsymbol{b}(x) \bmod q_0, \boldsymbol{a}(x)\boldsymbol{b}(x) \bmod q_1) =$
  $\left(\mathtt{NTT}_0^{-1}(\mathtt{NTT}_0(\boldsymbol{a}(x)) \cdot \mathtt{NTT}_0(\boldsymbol{b}(x))), \mathtt{NTT}_1^{-1}(\mathtt{NTT}_1(\boldsymbol{a}(x)) \cdot \mathtt{NTT}_1(\boldsymbol{b}(x)))\right)$



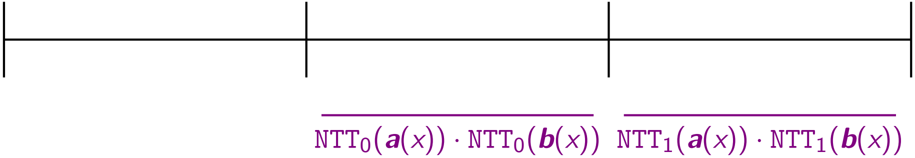
Stack Usage of 16-bit NTT-Based Polynomial Multiplications

- Each line segment $=$ 4096 bits, 12288 bits in total.
- A size-256 poly. of 16-bit coeffs. is stored in a segment.
- Compute
  $\boldsymbol{a}(x)\boldsymbol{b}(x) = \mathrm{CRT}\left(\mathtt{NTT}_0^{-1}(\mathtt{NTT}_0(\boldsymbol{a}(x)) \cdot \mathtt{NTT}_0(\boldsymbol{b}(x))), \mathtt{NTT}_1^{-1}(\mathtt{NTT}_1(\boldsymbol{a}(x)) \cdot \mathtt{NTT}_1(\boldsymbol{b}(x)))\right)$
- Notice $(\boldsymbol{a}(x)\boldsymbol{b}(x) \bmod q_0, \boldsymbol{a}(x)\boldsymbol{b}(x) \bmod q_1) =$
  $\left(\mathtt{NTT}_0^{-1}(\mathtt{NTT}_0(\boldsymbol{a}(x)) \cdot \mathtt{NTT}_0(\boldsymbol{b}(x))), \mathtt{NTT}_1^{-1}(\mathtt{NTT}_1(\boldsymbol{a}(x)) \cdot \mathtt{NTT}_1(\boldsymbol{b}(x)))\right)$



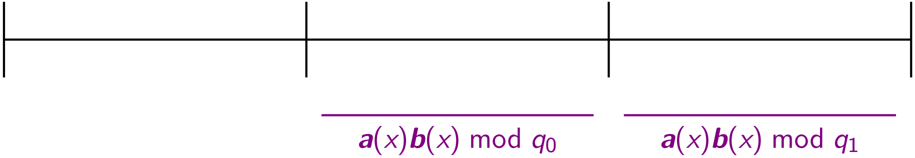
Stack Usage of 16-bit NTT-Based Polynomial Multiplications

- Each line segment = 4096 bits, 12288 bits in total.
- A size-256 poly. of 16-bit coeffs. is stored in a segment.
- Compute
  $\boldsymbol{a}(x)\boldsymbol{b}(x) = \text{CRT}\left(\mathtt{NTT}_0^{-1}(\mathtt{NTT}_0(\boldsymbol{a}(x)) \cdot \mathtt{NTT}_0(\boldsymbol{b}(x))), \mathtt{NTT}_1^{-1}(\mathtt{NTT}_1(\boldsymbol{a}(x)) \cdot \mathtt{NTT}_1(\boldsymbol{b}(x)))\right)$
- Notice $(\boldsymbol{a}(x)\boldsymbol{b}(x) \bmod q_0, \boldsymbol{a}(x)\boldsymbol{b}(x) \bmod q_1) =$
  $(\mathtt{NTT}_0^{-1}(\mathtt{NTT}_0(\boldsymbol{a}(x)) \cdot \mathtt{NTT}_0(\boldsymbol{b}(x))), \mathtt{NTT}_1^{-1}(\mathtt{NTT}_1(\boldsymbol{a}(x)) \cdot \mathtt{NTT}_1(\boldsymbol{b}(x))))$



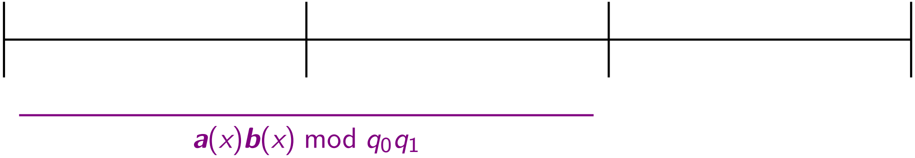
Stack Usage of 16-bit NTT-Based Polynomial Multiplications

- Each line segment = 4096 bits, 12288 bits in total.
- A size-256 poly. of 16-bit coeffs. is stored in a segment.
- Compute
  $\boldsymbol{a}(x)\boldsymbol{b}(x) = \mathrm{CRT}\left(\mathtt{NTT}_0^{-1}(\mathtt{NTT}_0(\boldsymbol{a}(x)) \cdot \mathtt{NTT}_0(\boldsymbol{b}(x))), \mathtt{NTT}_1^{-1}(\mathtt{NTT}_1(\boldsymbol{a}(x)) \cdot \mathtt{NTT}_1(\boldsymbol{b}(x)))\right)$
- Notice $(\boldsymbol{a}(x)\boldsymbol{b}(x) \bmod q_0, \boldsymbol{a}(x)\boldsymbol{b}(x) \bmod q_1) =$
  $(\mathtt{NTT}_0^{-1}(\mathtt{NTT}_0(\boldsymbol{a}(x)) \cdot \mathtt{NTT}_0(\boldsymbol{b}(x))), \mathtt{NTT}_1^{-1}(\mathtt{NTT}_1(\boldsymbol{a}(x)) \cdot \mathtt{NTT}_1(\boldsymbol{b}(x))))$



Stack Usage of 16-bit NTT-Based Polynomial Multiplications

- Each line segment = 4096 bits, 12288 bits in total.
- A size-256 poly. of 16-bit coeffs. is stored in a segment.
- Compute
  $\boldsymbol{a}(x)\boldsymbol{b}(x) = \text{CRT}\left(\texttt{NTT}_0^{-1}(\texttt{NTT}_0(\boldsymbol{a}(x)) \cdot \texttt{NTT}_0(\boldsymbol{b}(x))), \texttt{NTT}_1^{-1}(\texttt{NTT}_1(\boldsymbol{a}(x)) \cdot \texttt{NTT}_1(\boldsymbol{b}(x)))\right)$
- Notice $(\boldsymbol{a}(x)\boldsymbol{b}(x) \bmod q_0, \boldsymbol{a}(x)\boldsymbol{b}(x) \bmod q_1) =$
  $(\texttt{NTT}_0^{-1}(\texttt{NTT}_0(\boldsymbol{a}(x)) \cdot \texttt{NTT}_0(\boldsymbol{b}(x))), \texttt{NTT}_1^{-1}(\texttt{NTT}_1(\boldsymbol{a}(x)) \cdot \texttt{NTT}_1(\boldsymbol{b}(x))))$

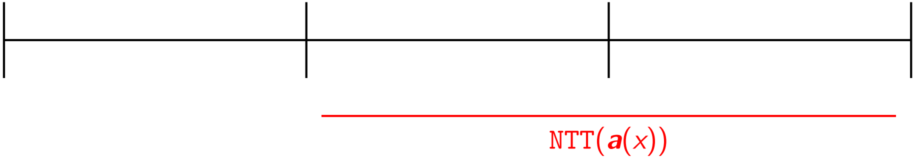


Our approach

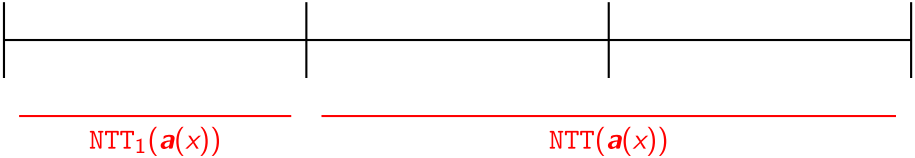
For a Cortex-M4, one 32-bit NTT is much faster than two 16-bit NTTs.

1. Start with 16-bit NTTs
2. Identify at which point that inevitably, corresponding elements in $\mathbb{Z}_{q_0}, \mathbb{Z}_{q_1}$ are *both* in memory
3. Replace operations in $\mathbb{Z}_{q_0}, \mathbb{Z}_{q_1}$ with $\mathbb{Z}_{q_0 q_1}$ for these elements

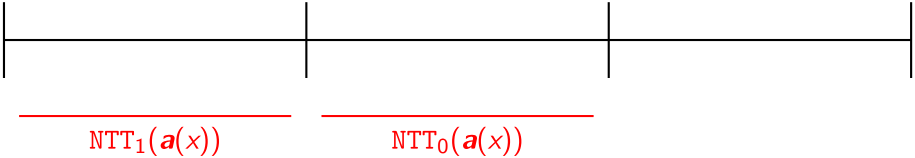
Combining 32-bit NTTs and 16-bit NTTs



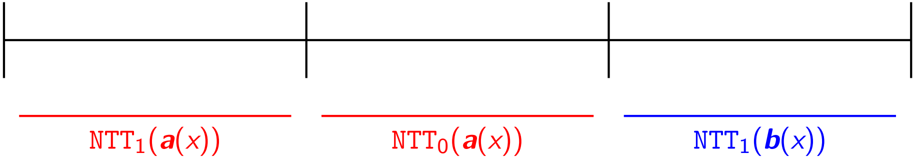
Combining 32-bit NTTs and 16-bit NTTs



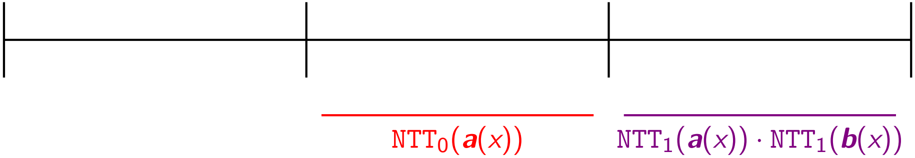
Combining 32-bit NTTs and 16-bit NTTs



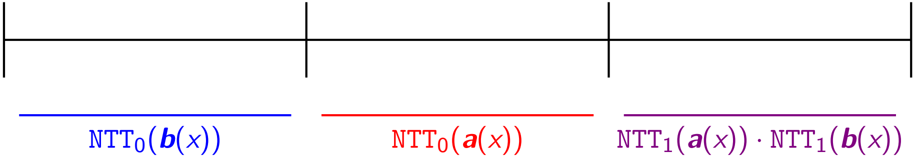
Combining 32-bit NTTs and 16-bit NTTs



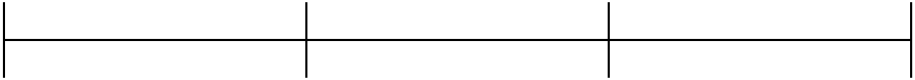
Combining 32-bit NTTs and 16-bit NTTs



Combining 32-bit NTTs and 16-bit NTTs

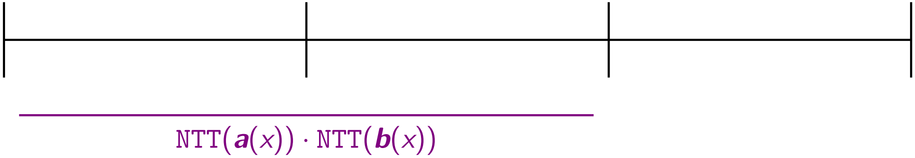


Combining 32-bit NTTs and 16-bit NTTs

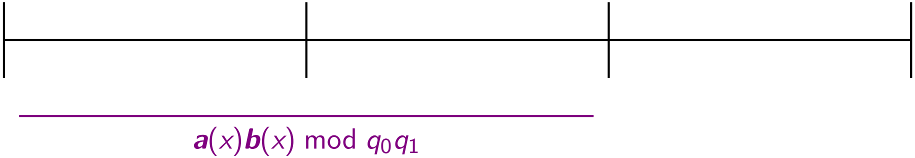


$\overline{\mathtt{NTT}_0(\boldsymbol{a}(x)) \cdot \mathtt{NTT}_0(\boldsymbol{b}(x))}$ $\overline{\mathtt{NTT}_1(\boldsymbol{a}(x)) \cdot \mathtt{NTT}_1(\boldsymbol{b}(x))}$

Combining 32-bit NTTs and 16-bit NTTs



Combining 32-bit NTTs and 16-bit NTTs



Performance of NTT-Based Polynomial Multiplications on Cortex-M4

Table 1: NTT–related functions on Cortex-M4. Numbers of the last two columns are extracted from paper.

	32-bit	16-bit + 16-bit	32-bit	16-bit
NTT	5 853	4 374+ 4 822	5853	4822
NTT^{-1}	7 137	–	7137	4817
base_mul	–	3731 + 2 965	4186	2965
$\bmod p_i$	–	0 + 1 171	-	-
CRT	–	2 435	-	2 435
poly_mul	32 488		23 029	37 287

Strategies for `MatrixVectorMul`

- Strategy A: $A^T s = \mathtt{NTT}^{-1}(\mathtt{NTT}(A^T) \cdot \underline{\mathtt{NTT}(s)})$
- Strategy B: $A_{i,j}^T s_j = \mathtt{NTT}^{-1}(\mathtt{NTT}(A_{i,j}^T) \cdot \underline{\mathtt{NTT}(s_j)})$
- Strategy C: $A^T s = \mathtt{NTT}^{-1}(\mathtt{NTT}(A^T) \cdot \mathtt{NTT}(s))$
- Strategy D: $A_{i,j}^T s_j = \mathtt{NTT}^{-1}(\mathtt{NTT}(A_{i,j}^T) \cdot \mathtt{NTT}(s_j))$

Figure 1: Strategies for `MatrixVectorMul`.

	Cache $\mathtt{NTT}(s)$	Compute $\mathtt{NTT}(s)$
Acc. in NTT domain	A	C
Acc. in $\mathbb{Z}_{8192}[x]$	B	D

- Key generation, $A^T s$: strategies A, B, D
- Encryption, As': strategies A, C, D

First–Order Masked `MatrixVectorMul` and `InnerProd`

First–Order Masked `MatrixVectorMul` and `InnerProd`

- Split $s' = s'_0 + s'_1$ (first-order)
- Compute (As'_0, As'_1)
- $(As'_0, As'_1) = (\texttt{NTT}^{-1}(\texttt{NTT}(A) \cdot \texttt{NTT}(s'_0)), \texttt{NTT}^{-1}(\texttt{NTT}(A) \cdot \texttt{NTT}(s'_1)))$
  - $l^2 + 2l$ NTTs
  - $2l$ $\texttt{NTT}^{-1}$s
- Coefficient rings of s'_0, s'_1: $\mathbb{Z}_{8192}$ instead of $\left\{-\frac{\mu}{2}, \dots, \frac{\mu}{2}\right\}$
  - Compute with one 32-bit NTT and one 16-bit NTT
- In total:
  - $l^2 + 2l$ 32-bit NTTs
  - $l^2 + 2l$ 16-bit NTTs
  - $2l$ 32-bit $\texttt{NTT}^{-1}$s
  - $2l$ 16-bit $\texttt{NTT}^{-1}$s

Saber on Cortex-M3

Differences Between Cortex-M3 and Cortex-M4

- No floating-point registers
- No DSP extension (`s{mul, mla}{b, t}{b, t}`, `smlad{, x}`, `{u, s}{add, sub}{8, 16}`)
  - 16-bit NTTs are much slower
- `{u, s}{mul, mla}l` takes input-dependent cycles
  - `NTT_leak`: 32-bit NTTs are variable time (for public data)
  - Constant-time NTTs: Emulate 32-bit NTTs with `mul`, `mla`, ... [GKS21] (much slower)
- Question: which is better?
  - Cortex-M4: one 32-bit NTT is faster than two 16-bit NTTs
  - Cortex-M3: two 16-bit NTTs vs one 32-bit NTT

Saber on Cortex-M3 i

- 16-bit NTTs only
  - $As' = \texttt{NTT}^{-1}(\texttt{NTT}(A) \cdot \texttt{NTT}(s'))$ where $\texttt{NTT}/\texttt{NTT}^{-1}$ is a pair of 16-bit NTT/iNTTs
- 32-bit NTTs only
  - $As' = \texttt{NTT}^{-1}(\texttt{NTT\_leak}(A) \cdot \texttt{NTT}(s'))$ where $\texttt{NTT}$ is the constant-time NTT.
- 32-bit NTTs and 16-bit NTT
  - $As' = \texttt{NTT}^{-1}((a \mapsto (a \bmod q_0, a \bmod q_1) \circ \texttt{NTT\_leak})(A) \cdot \texttt{NTT}(s'))$ where $\texttt{NTT}/\texttt{NTT}^{-1}$ is a pair of 16-bit NTT/iNTTs
  - Doesn't worth it as $a \mapsto (a \bmod q_0, a \bmod q_1) \approx \texttt{NTT} - \texttt{NTT\_leak}$

Saber on Cortex-M3 ii

Table 2: NTT–related functions on Cortex-M3.

	2 × 16-bit	32-bit
NTT	16 774	31 056
NTT_leak	–	19 363
NTT^{-1}	19 079	37 394
base_mul	11 933	8 532
$\bmod p_i$	–	–
CRT	4 642	–
poly_mul	69 202	96 345

Results

Cortex-M4 Results i

Table 3: Unprotected Saber on Cortex-M4.

			LightSaber		Saber		FireSaber	
			cc	stack	cc	stack	cc	stack
M4	[MKV20] (stack)	K	612k	3 564	1 230k	4 348	2 046k	5 116
		E	880k	**3 148**	1 616k	3 412	2 538k	3 668
		D	976k	**3 164**	1 759k	3 420	2 740k	3 684
	[CHK+21] (speed)	K	360k	14 604	658k	23 284	1 008k	37 116
		E	513k	16 252	864k	32 620	1 255k	40 484
		D	498k	16 996	835k	33 824	1 227k	41 964
	This work 32-bit (speed, A)	K	**353k**	5 764	**644k**	6 788	**990k**	7 812
		E	**487k**	6 444	**826k**	7 468	**1 208k**	8 484
		D	**456k**	6 440	**777k**	7 484	**1 152k**	8 500
	This work hybrid (stack, D)	K	423k	**3 428**	819k	**3 940**	1 315k	**4 452**
		E	597k	3 204	1 063k	**3 332**	1 617k	**3 468**
		D	583k	3 220	1 039k	**3 348**	1 594k	**3 484**

Cortex-M4 Results ii

Table 4: Masked Saber ($l = 3$) on the Cortex-M4.

	Decapsulation	
	cc	stack
[VBDK+20]	2 833k	11 656
This work (speed, A)	**2 385k**	16 140
This work (C)	2 615k	10 476
This work (stack, D)	2 846k	**8 432**

Table 5: Masking cycles/stack overhead.

	unmasked A		unmasked D	
	cc	stack	cc	stack
masked A	3.07	2.16	2.30	4.82
masked C	3.37	1.40	2.52	3.13
masked D	3.66	1.13	2.74	2.52

Cortex-M3 Results

Table 6: Unprotected Saber on Cortex-M3.

			LightSaber		Saber		FireSaber	
			cc	stack	cc	stack	cc	stack
M3	pqm3 Toom (speed)	K	710k	9 652	1 328k	13 252	2 171k	20 116
		E	967k	11 372	1 738k	15 516	2 688k	22 964
		D	1 081k	12 116	1 902k	16 612	2 933k	24 444
	This work 16-bit (speed, A)	K	**540k**	5 756	**939k**	6 788	**1 439k**	7 812
		E	**715k**	6 436	**1 194k**	7 468	**1 751k**	8 492
		D	**749k**	6 436	**1 237k**	7 468	**1 811k**	8 492
	This work 16-bit (stack, D)	K	632k	**3 420**	1 253k	**3 932**	1 955k	**4 444**
		E	887k	**3 204**	1 614k	**3 332**	2 427k	**3 460**
		D	923k	**3 204**	1 657k	**3 332**	2 487k	**3 460**
	This work 32-bit (speed, A)	K	594k	5 732	1 057k	6 756	1 553k	7 788
		E	800k	6 412	1 330k	7 444	1 883k	8 468
		D	877k	6 420	1 429k	7 452	2 016k	8 476

Summary

- Cortex-M4:
  - Cycles: NTT (speed) $<<$ NTT (stack) $\approx$ non-NTT (speed: TMVP, Toom–Cook) $<<$ non-NTT (stack: Karatsuba)
  - Stack: NTT (stack) $\approx$ non-NTT (stack) $<$ non-NTT (speed) $<$ NTT (speed)
- Cortex-M3:
  - Cycles: 16-bit NTT (speed) $<$ 32-bit NTT (speed) $<$ 16-bit NTT (stack) $<$ non-NTT (speed, Toom–Cook)
  - Stack: 16-bit NTT (stack) $<$ 32-bit NTT (speed) $\approx$ 16-bit NTT (speed) $<$ non-NTT (speed, Toom–Cook)

Thank you for your attention

Reference i

RC Agarwal and C Burrus.
Fast convolution using Fermat number transforms with applications to digital filtering.
IEEE Transactions on Acoustics, Speech, and Signal Processing, 22(2):87–97, 1974.

Chi-Ming Marvin Chung, Vincent Hwang, Matthias J. Kannwischer, Gregor Seiler, Cheng-Jhih Shih, and Bo-Yin Yang.
NTT Multiplication for NTT-unfriendly Rings New Speed Records for Saber and NTRU on Cortex-M4 and AVX2.
IACR Transactions on Cryptographic Hardware and Embedded Systems, 2021(2):159–188, 2021.
https://tches.iacr.org/index.php/TCHES/article/view/8791.

Reference ii

Denisa O. C. Greconici, Matthias J. Kannwischer, and Daan Sprenkels.
Compact Dilithium Implementations on Cortex-M3 and Cortex-M4.
IACR Transactions on Cryptographic Hardware and Embedded Systems, 2021(1):1–24, 2021.
https://tches.iacr.org/index.php/TCHES/article/view/8725.

Jose Maria Bermudo Mera, Angshuman Karmakar, and Ingrid Verbauwhede.
Time-memory trade-off in Toom-Cook multiplication: an application to module-lattice based cryptography.
IACR Transactions on Cryptographic Hardware and Embedded Systems, 2020(2):222–244, 2020.
https://tches.iacr.org/index.php/TCHES/article/view/8550.

Reference iii

Michiel Van Beirendonck, Jan-Pieter D'Anvers, Angshuman Karmakar, Josep Balasch, and Ingrid Verbauwhede.
A side-channel resistant implementation of SABER.
IACR Cryptol. ePrint Arch., 2020:733, 2020.
https://eprint.iacr.org/2020/733.