KU LEUVEN

Time-memory trade-off in Toom-Cook multiplication: an application to module-lattice based cryptography

Jose Maria Bermudo Mera
Angshuman Karmakar
Ingrid Verbauwhede

CHES 2020

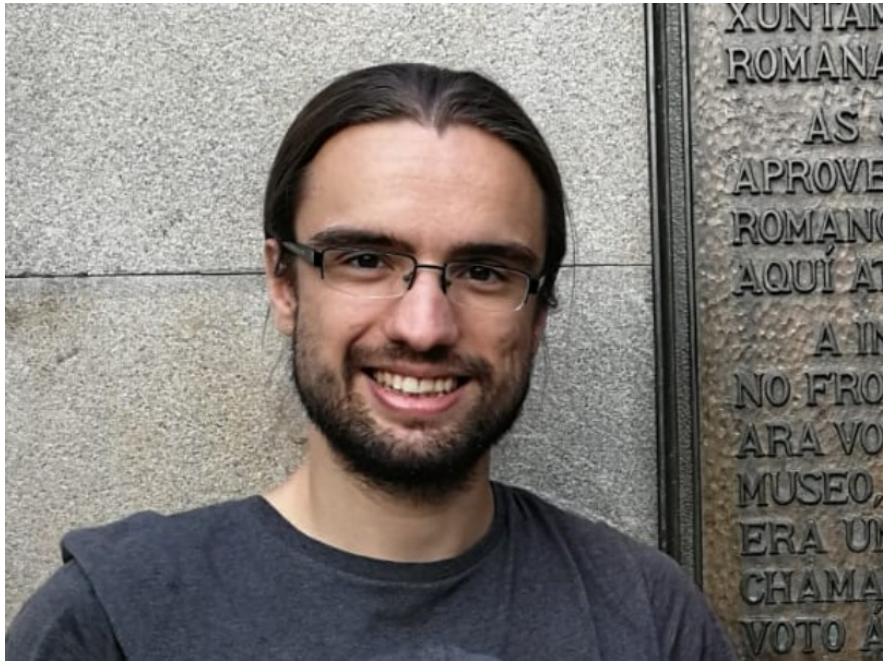
FENTEC

COSIC

erc
European Research Council

Speaker

- Jose Maria Bermudo Mera
- PhD researcher at KU Leuven
  - Implementation aspects of lattice-based cryptography
  - Publications at https://bit.ly/311onEM
- Contact me at Jose.Bermudo@esat.kuleuven.be in EN/ES/IT



Contents

- Introduction
- Speed optimizations
- Memory optimizations
- Results
- Conclusions

Context

- Security of existing public key schemes is based on hard problems (e.g., integer factorization and discrete logarithm)
  - Shor's algorithm solves them *efficiently* on quantum computers
- NIST has recently announced the finalists of the standardization contest for post-quantum cryptography:

KEM		**Signature**	
McEliece	Code-based	Dilithium	Lattice-based
Kyber	Lattice-based	Falcon	Lattice-based
NTRU	Lattice-based	Rainbow	Multivariate
Saber	Lattice-based		

Context

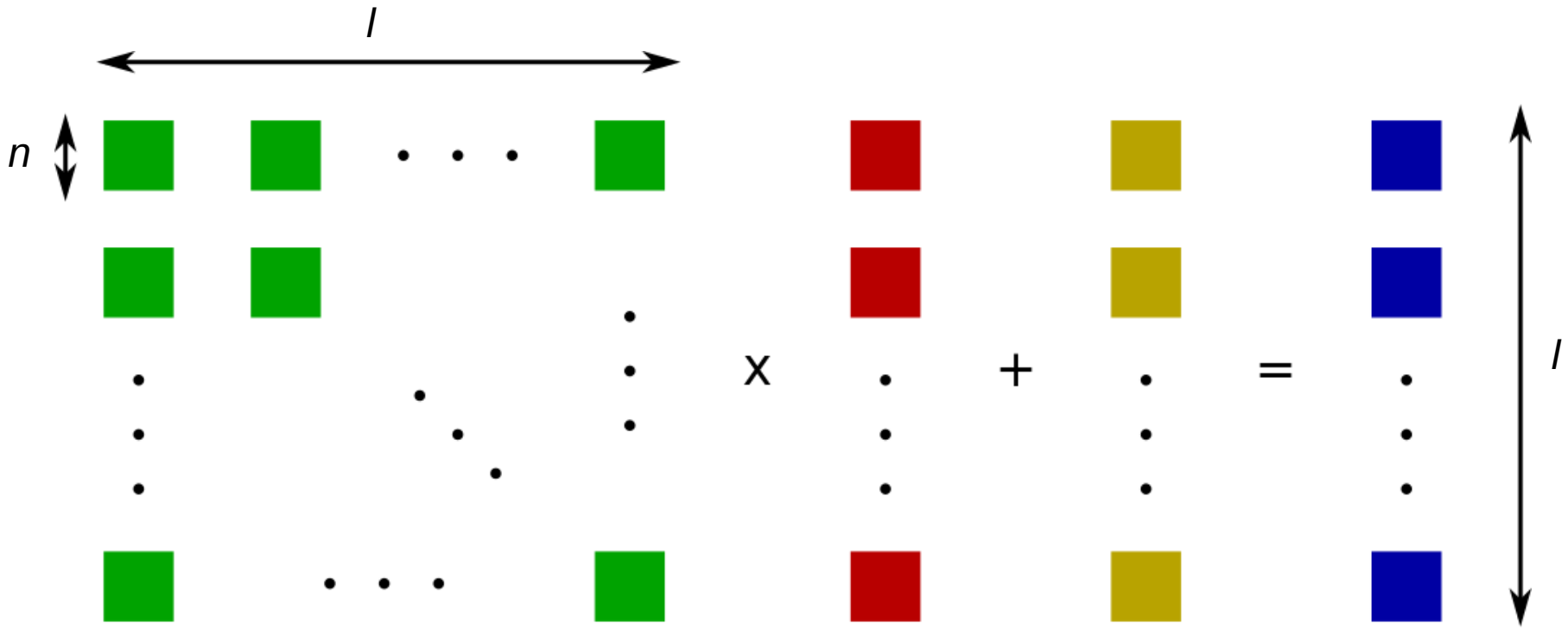
- Security of existing public key schemes is based on hard problems (e.g., integer factorization and discrete logarithm)
  - Shor's algorithm solves them *efficiently* on quantum computers
- NIST has recently announced the finalists of the standardization contest for post-quantum cryptography:

KEM	
McEliece	Code-based
Kyber	Lattice-based
NTRU	Lattice-based
Saber	Lattice-based

Signature	
Dilithium	Lattice-based
Falcon	Lattice-based
Rainbow	Multivariate

Module lattices

Module lattices



Module lattices → Polynomial arithmetic rather than matrix-vector multiplication

Polynomial multiplication – coefficient form

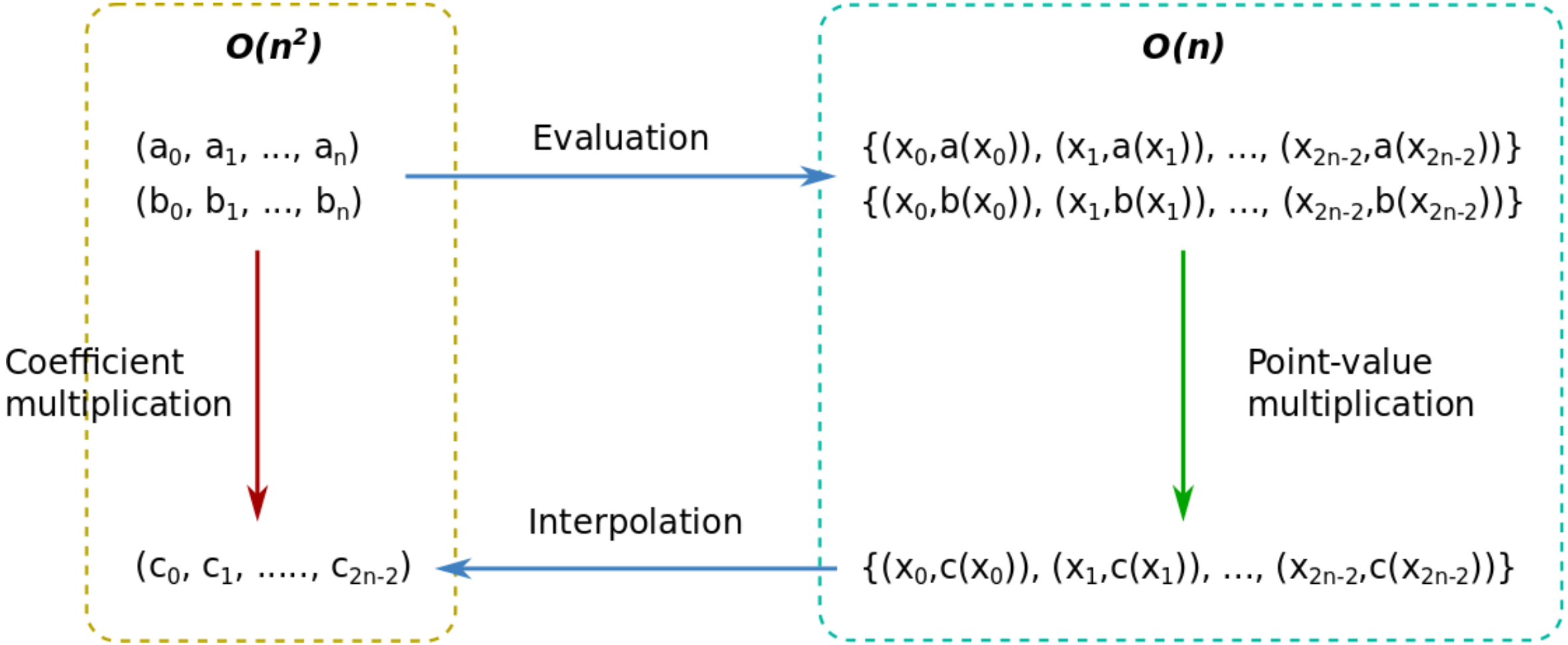
$A(x) = a_3x^3 + a_2x^2 + a_1x + a_0$

$B(x) = b_3x^3 + b_2x^2 + b_1x + b_0$

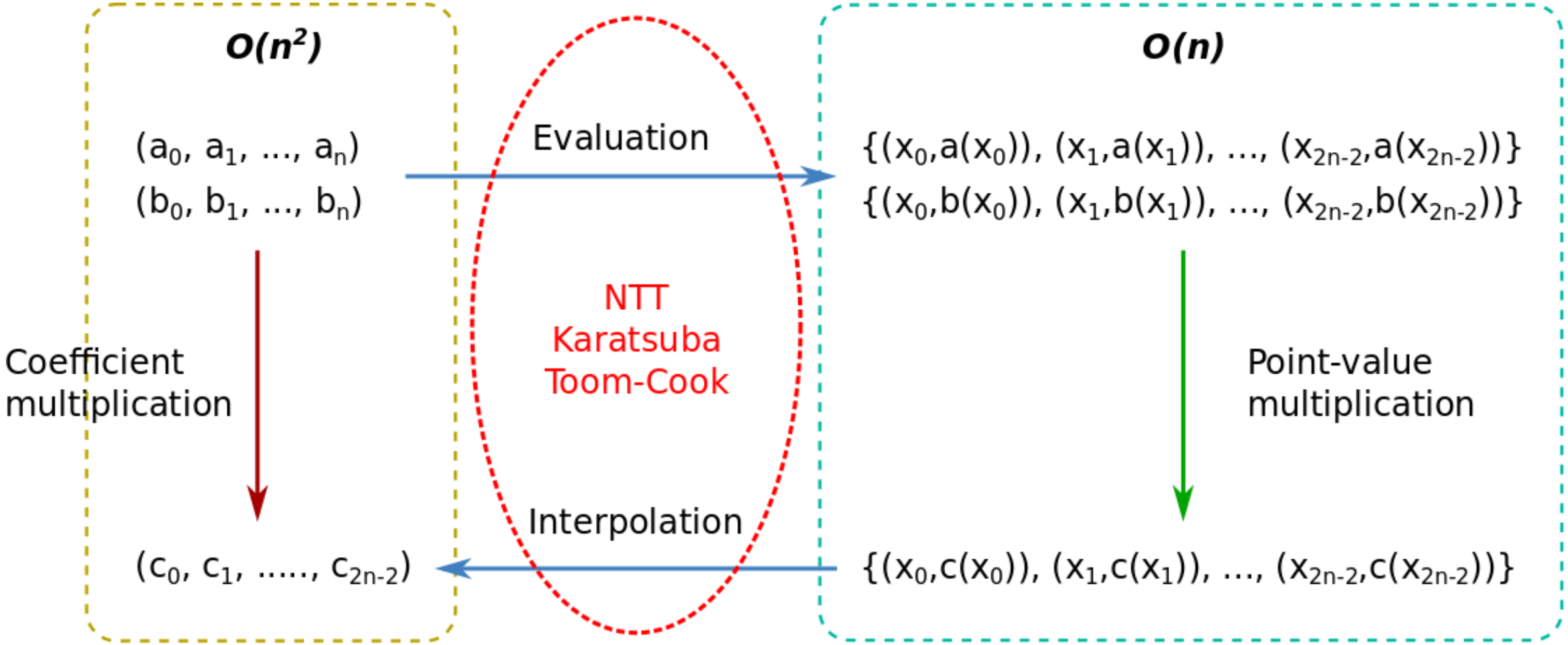
			a_3	a_2	a_1	a_0
		x	b_3	b_2	b_1	b_0
	0	0	0	$a_0 \cdot b_3$	$a_0 \cdot b_2$	$a_0 \cdot b_1$	$a_0 \cdot b_0$
+	0	0	$a_1 \cdot b_3$	$a_1 \cdot b_2$	$a_1 \cdot b_1$	$a_1 \cdot b_0$	0
	0	$a_2 \cdot b_3$	$a_2 \cdot b_2$	$a_2 \cdot b_1$	$a_2 \cdot b_0$	0	0
	$a_3 \cdot b_3$	$a_3 \cdot b_2$	$a_3 \cdot b_1$	$a_3 \cdot b_0$	0	0	0
	c_6	c_5	c_4	c_3	c_2	c_1	c_0

$C(x) = c_6x^6 + c_5x^5 + c_4x^4 + c_3x^3 + c_2x^2 + c_1x + c_0$

Polynomial multiplication – point-value form



Polynomial multiplication – point-value form

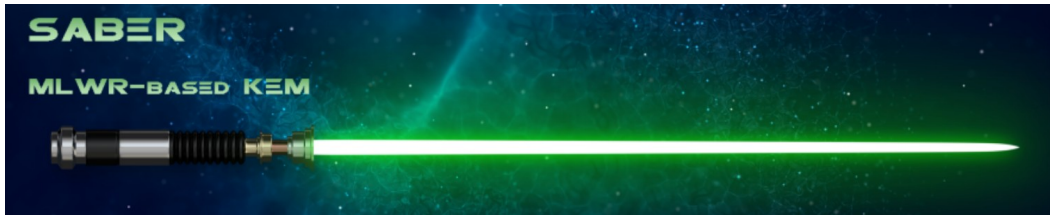


Evaluation/interpolation

- NTT
  - Complexity $O(n \log n)$
  - Requires prime modulus q and $q \equiv 1 \bmod 2n$
- Karatsuba
  - Complexity $O(n^{1.585})$
  - No restrictions
- Toom-Cook k-way
  - Complexity $O(c(k)\, n^{\log(2k-1)/\log(k)})$
  - No restrictions

Evaluation/interpolation

- NTT
  - Complexity $O(n \log n)$
  - Requires prime modulus q and $q \equiv 1 \bmod 2n$
- Karatsuba
  - Complexity $O(n^{1.585})$
  - No restrictions
- Toom-Cook k-way
  - Complexity $O(c(k)\, n^{\log(2k-1)/\log(k)})$
  - No restrictions



Saber *l* = 3

- 12 polynomial multiplication per encryption

Matrix multiplication for the ciphertext

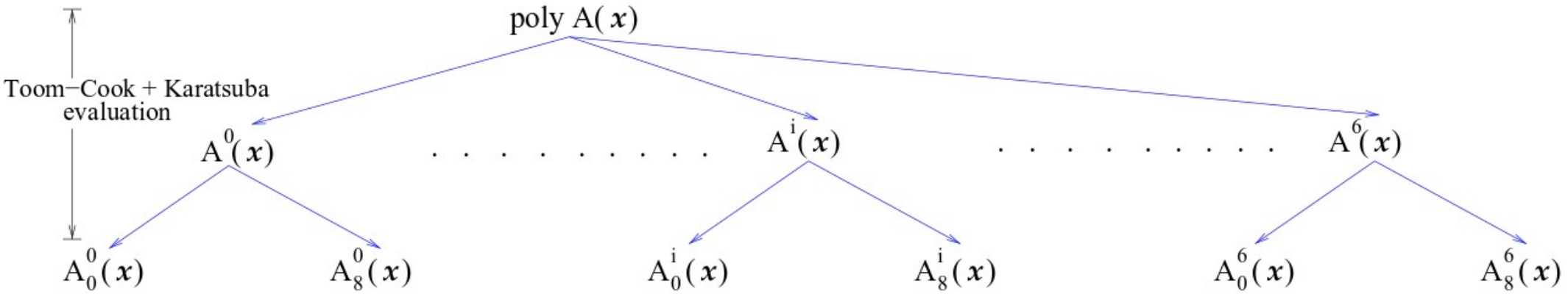
$$\begin{pmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{pmatrix} \cdot \begin{pmatrix} s_0 \\ s_1 \\ s_2 \end{pmatrix}$$

Vector multiplication for the key

$$\begin{pmatrix} b_0 \\ b_1 \\ b_2 \end{pmatrix}^T \cdot \begin{pmatrix} s'_0 \\ s'_1 \\ s'_2 \end{pmatrix}$$

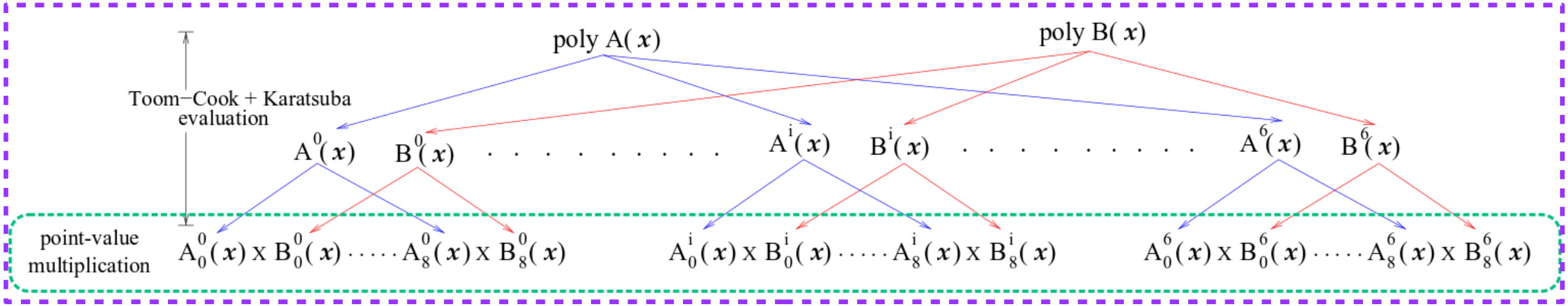
- Algorithmic choice for polymul
  - Top layer: Toom-Cook 4-way (**1 256x256 to 7 64x64**)
  - Intermediate layer: 2 levels of Karatsuba (**1 64x64 to 9 16x16**)
  - Bottom layer: 16x16 coefficient multiplication (**63 16x16 in total**)

Polynomial multiplication in Saber

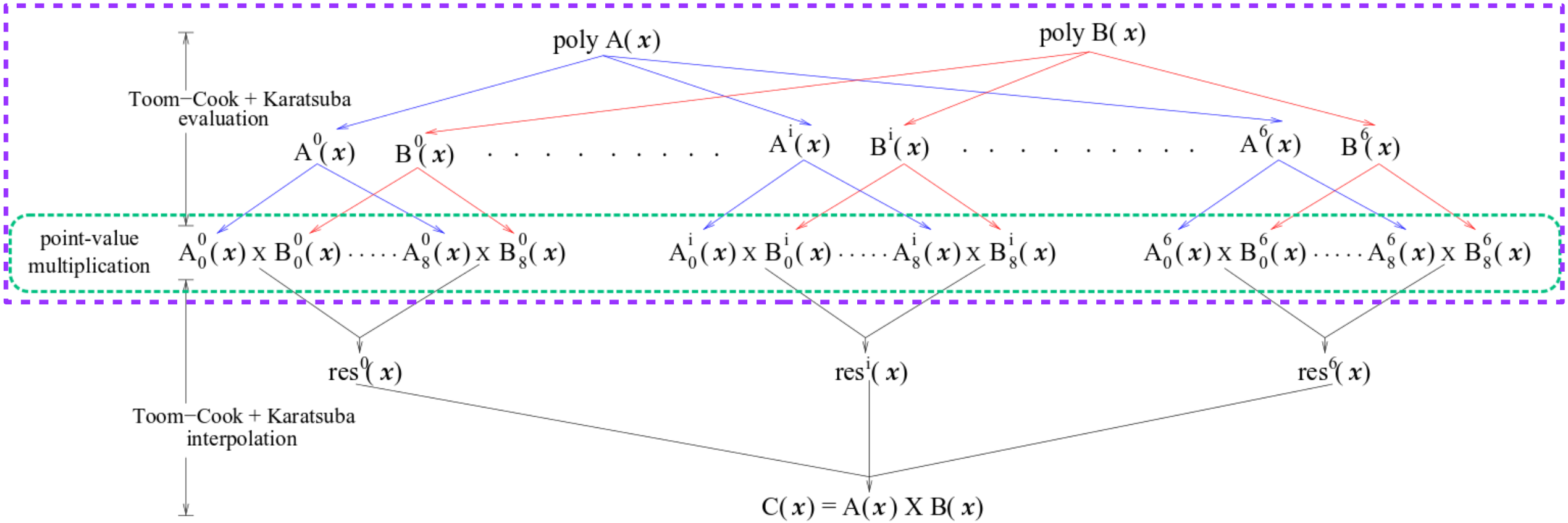


KU LEUVEN

Polynomial multiplication in Saber



Polynomial multiplication in Saber



KU LEUVEN

Speed optimizations

Lazy interpolation

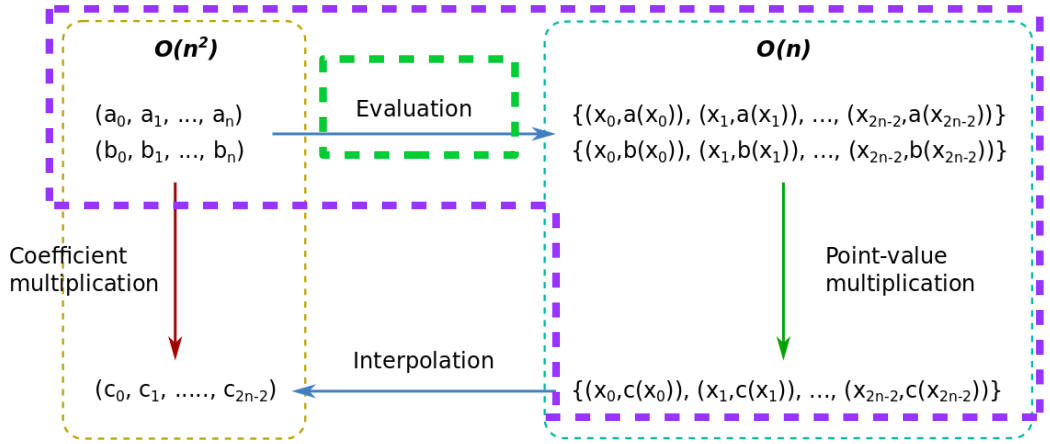
$$\begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{bmatrix} \cdot \begin{bmatrix} s_0 \\ s_1 \\ s_2 \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \\ b_2 \end{bmatrix}$$

$$= \begin{bmatrix} a_{00} \cdot s_0 + a_{01} \cdot s_1 + a_{02} \cdot s_2 \\ a_{10} \cdot s_0 + a_{11} \cdot s_1 + a_{12} \cdot s_2 \\ a_{20} \cdot s_0 + a_{21} \cdot s_1 + a_{22} \cdot s_2 \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \\ b_2 \end{bmatrix}$$

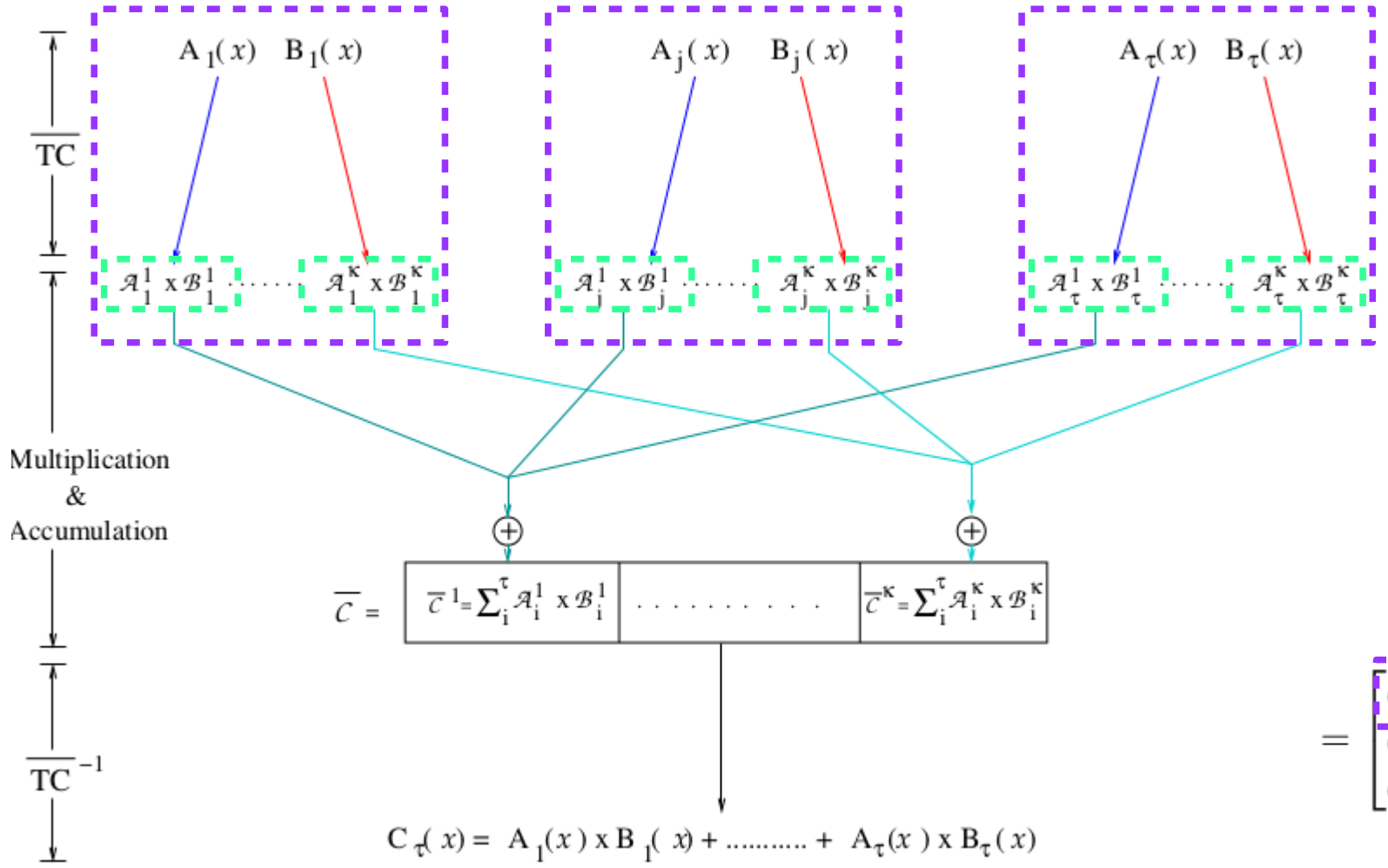
Pre-computation

$$\begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{bmatrix} \cdot \begin{bmatrix} s_0 \\ s_1 \\ s_2 \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \\ b_2 \end{bmatrix}$$

$$= \begin{bmatrix} a_{00} \cdot s_0 + a_{01} \cdot s_1 + a_{02} \cdot s_2 \\ a_{10} \cdot s_0 + a_{11} \cdot s_1 + a_{12} \cdot s_2 \\ a_{20} \cdot s_0 + a_{21} \cdot s_1 + a_{22} \cdot s_2 \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \\ b_2 \end{bmatrix}$$



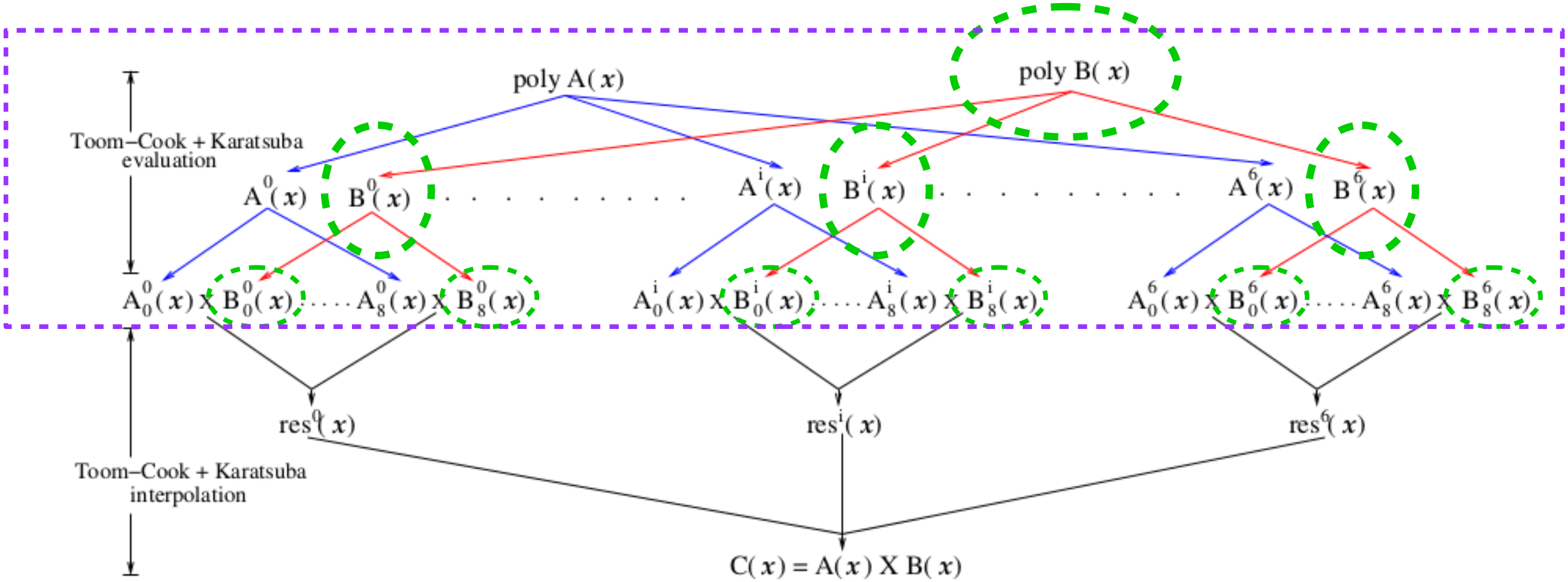
Lazy interpolation



$$\begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{bmatrix} \cdot \begin{bmatrix} s_0 \\ s_1 \\ s_2 \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \\ b_2 \end{bmatrix}$$

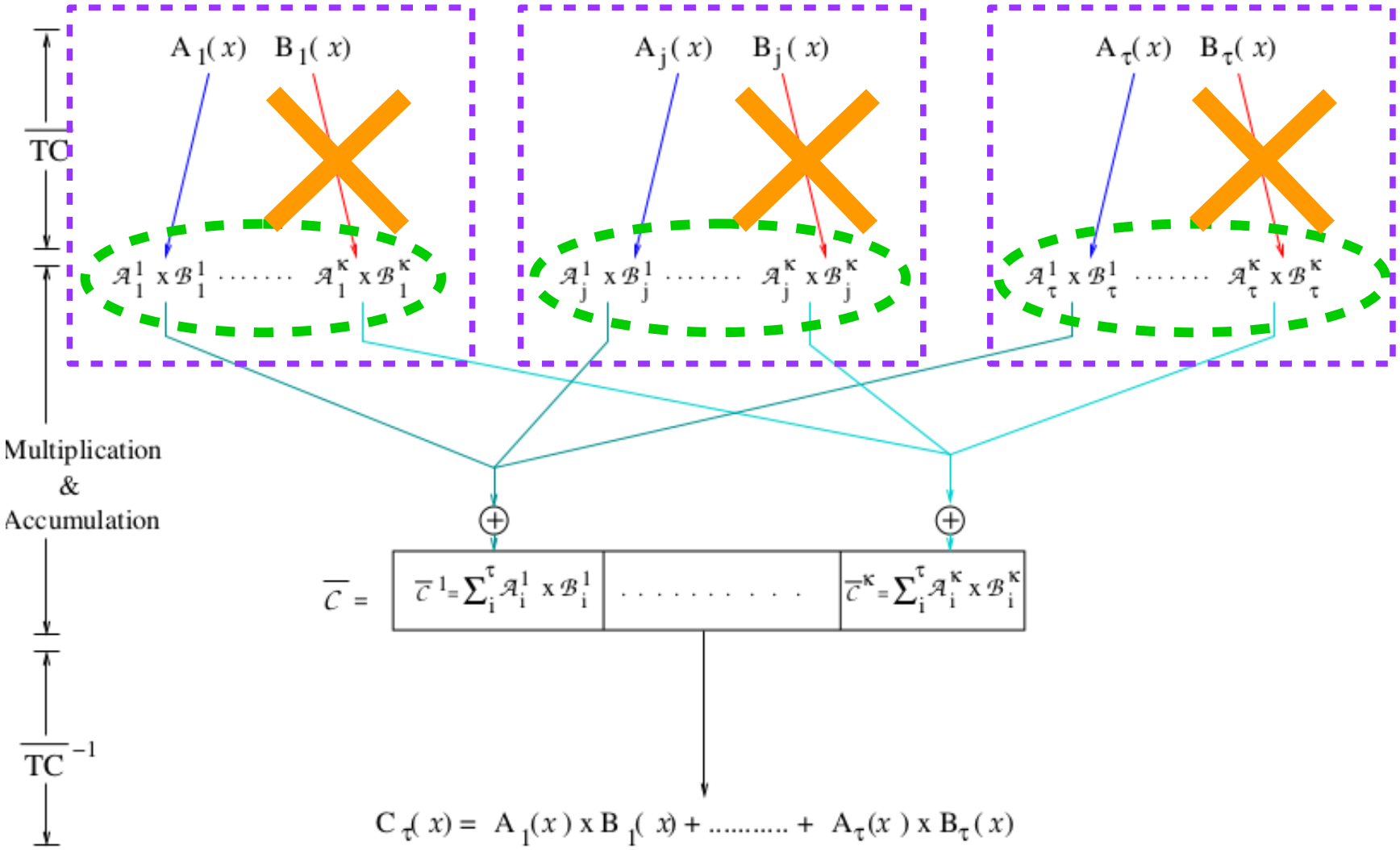
$$= \begin{bmatrix} a_{00} \cdot s_0 + a_{01} \cdot s_1 + a_{02} \cdot s_2 \\ a_{10} \cdot s_0 + a_{11} \cdot s_1 + a_{12} \cdot s_2 \\ a_{20} \cdot s_0 + a_{21} \cdot s_1 + a_{22} \cdot s_2 \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \\ b_2 \end{bmatrix}$$

Precomputation



KU LEUVEN

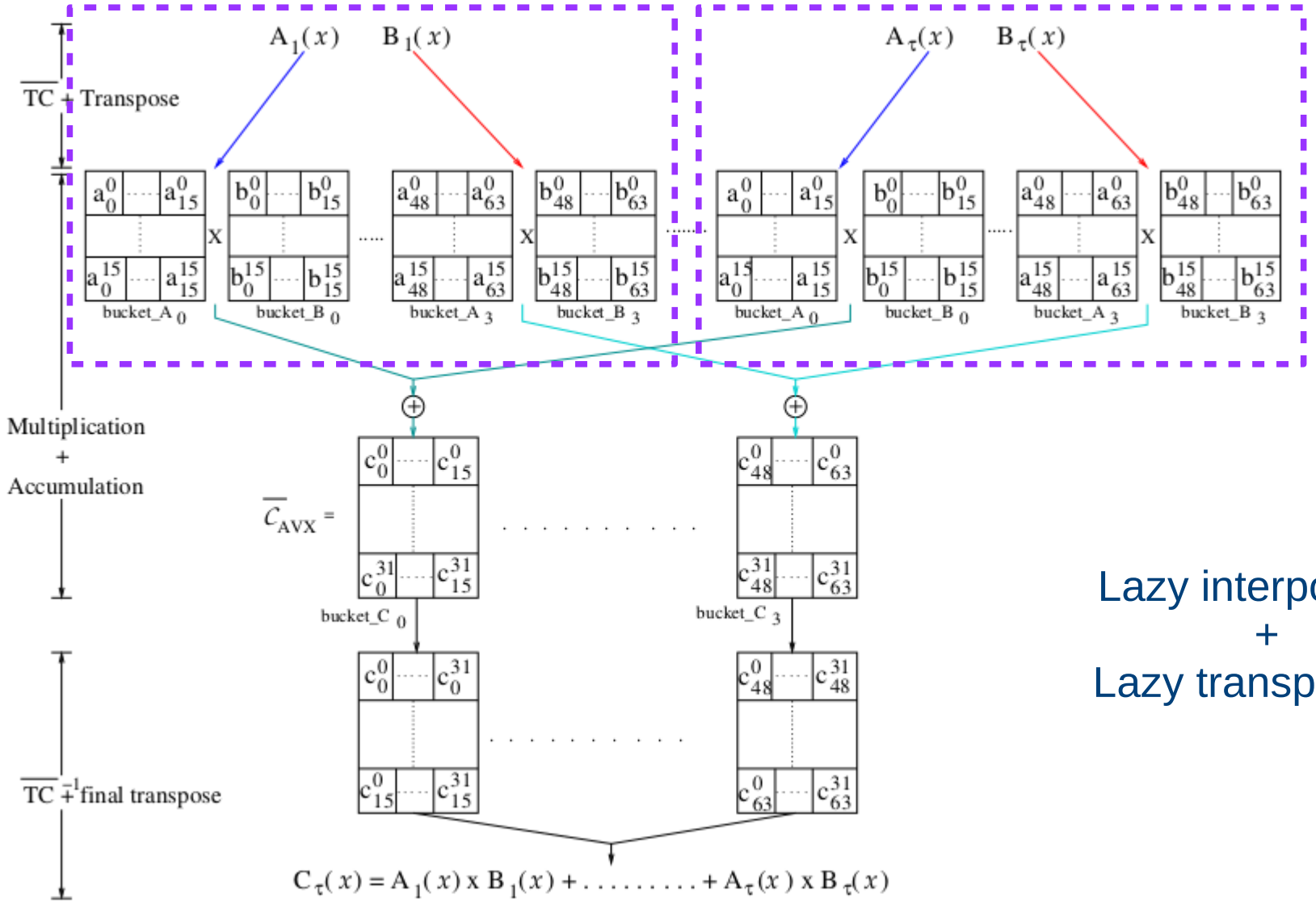
Lazy interpolation + Precomputation



Analysis of our improvements

Primitives	Polynomial multiplications	Evaluations		Interpolations	
		Base case	Our work	Base case	Our work
KeyGen	l^2	$2l^2$	$l^2 + l$	l^2	l
Encryption	$l^2 + l$	$2(l^2 + l)$	$l^2 + 2l$	$l^2 + l$	$l + 1$
Decryption	l	$2l$	$2l$	l	1

Implementation on AVX2



Lazy interpolation
+
Lazy transposition

KU LEUVEN

Implementation on Cortex-M4

Original schoolbook

```
1.   ldr r6, [r1, #0]
2.   ldr.w ip, [r1, #4]
3.   ldr.w r3, [r1, #8]
4.   ldr.w sl, [r1, #12]
5.   ldr.w r7, [r2, #0]
6.   ldr.w r8, [r2, #4]
7.   ldr.w r4, [r2, #8]
8.   ldr.w lr, [r2, #12]
9.   smulbb r9, r7, r6
10.  smuadx fp, r7, r6
11.  pkhbt r9, r9, fp, lsl #16
12.  str.w r9, [r0]
13.  smuadx fp, r7, ip
14.  smulbb r5, r7, ip
15.  pkhbt r9, r8, r7
16.  smladx fp, r8, r6, fp
17.  smlad r5, r9, r6, r5
18.  pkhbt fp, r5, fp, lsl #16
19.  str.w fp, [r0, #4]
```

Proposed modification

```
1.   ldr.w r6, [r1, #0]
2.   ldr.w ip, [r1, #4]
3.   ldr.w r3, [r1, #8]
4.   ldr.w sl, [r1, #12]
5.   ldrh.w r9, [r2]
6.   ldrh.w fp, [r2, #2]
7.   ldr.w r7, [r0, #0]
8.   ldr.w r8, [r0, #4]
9.   ldr.w r4, [r0, #8]
10.  ldr.w lr, [r0, #12]
11.  smlabb r9, r7, r6, r9
12.  smladx fp, r7, r6, fp
13.  pkhbt r9, r9, fp, lsl #16
14.  ldrh.w fp, [r2, #6]
15.  ldrh.w r5, [r2, #4]
16.  str.w r9, [r2]
17.  smladx fp, r7, ip, fp
18.  smlabb r5, r7, ip, r5
19.  pkhbt r9, r8, r7
20.  smladx fp, r8, r6, fp
21.  smlad r5, r9, r6, r5
22.  pkhbt fp, r5, fp, lsl #16
23.  str.w fp, [r2, #4]
```

pre-load

accumulate

pre-load

accumulate

Small storage for secrets

- Secrets are stored as polynomials with $n = 256$ coefficients mod $q = 2^{13}$
- Secrets are sampled from a centered binomial distribution
  - $\beta_\mu \rightarrow$ coefficients lie in $[-\mu, \mu]$
  - Worst case for Saber $\mu = 5$
- Instead, store secrets using only 4 bits per coefficient

Advantages

- Reduced footprint of the secret keys

	Old	This work	**Compression**
$l = 2$	1568	992	**36.7%**
$l = 3$	2304	1440	**37.5%**
$l = 4$	3040	1888	**38.2%**

- Simple packing/unpacking functions
  - Embed unpacking in multiplication evaluation

Memory optimizations

- Book-keeping of randomness for hash functions
- Just-in-time polynomial generation
- In-place verification of ciphertext
- Use only Karatsuba for multiplication
- Merge unpacking of secrets and Karatsuba evaluation

Results – Matrix-vector multiplication

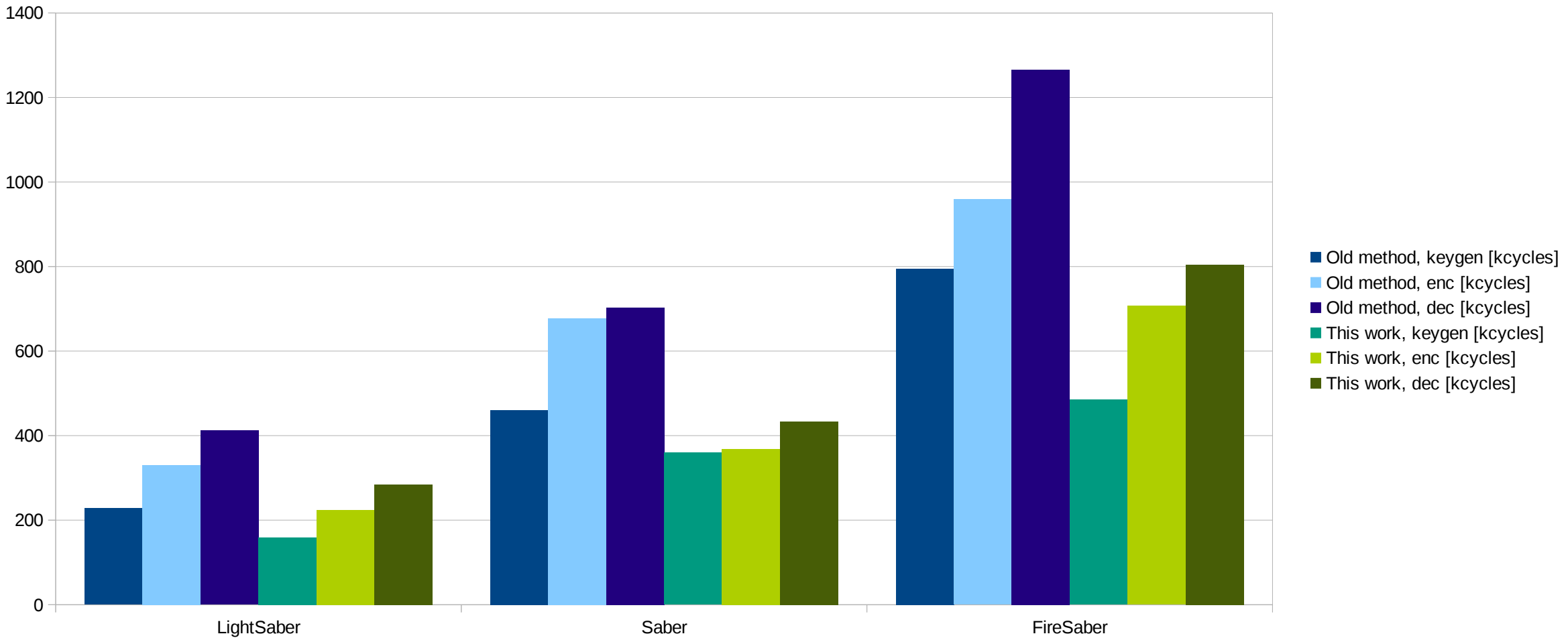
- AVX2

	Old method	This work	**Speedup**
$l = 2$	10199	7214	**29.3%**
$l = 3$	21356	13574	**36.4%**
$l = 4$	39039	24767	**36.6%**

- Cortex-M4

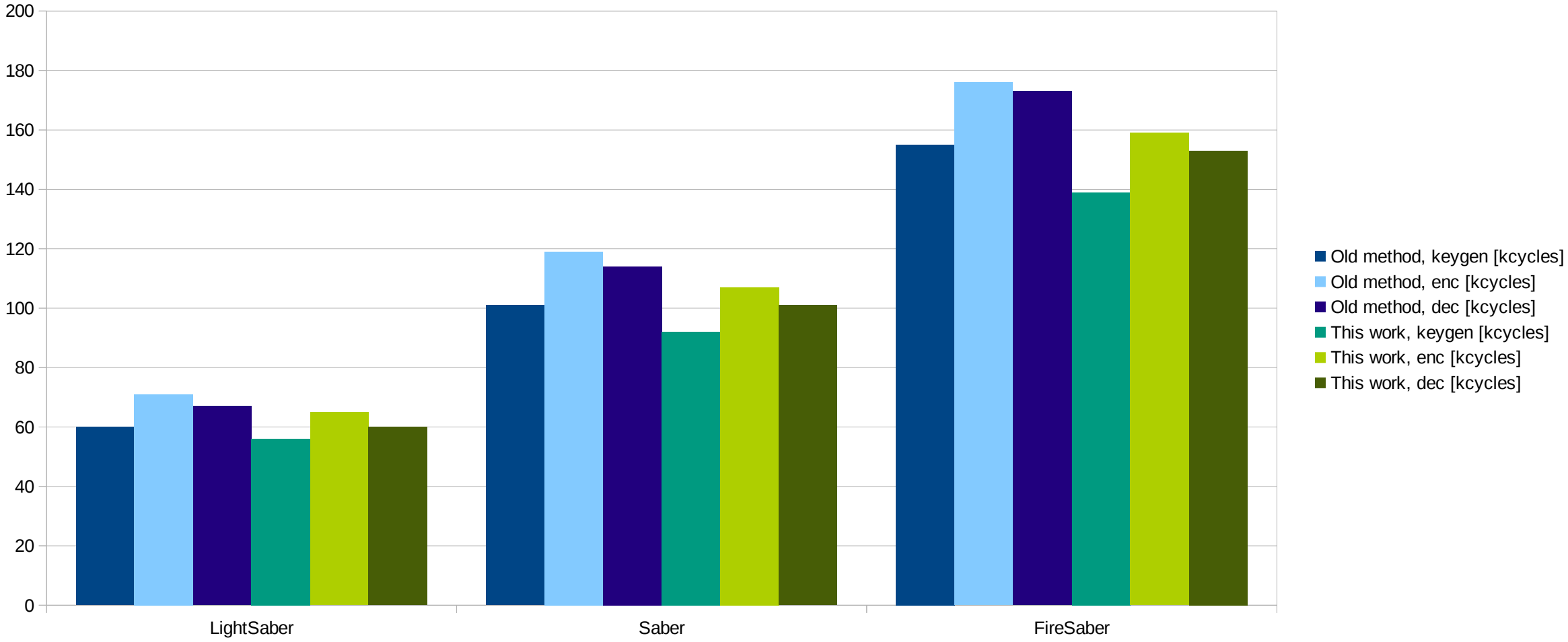
	Old method	This work	**Speedup**
$l = 2$	162 $k$cycles	159 $k$cycles	**1.9%**
$l = 3$	361 $k$cycles	317 $k$cycles	**12.2%**
$l = 4$	646 $k$cycles	528 $k$cycles	**18.3%**

Results – plain C



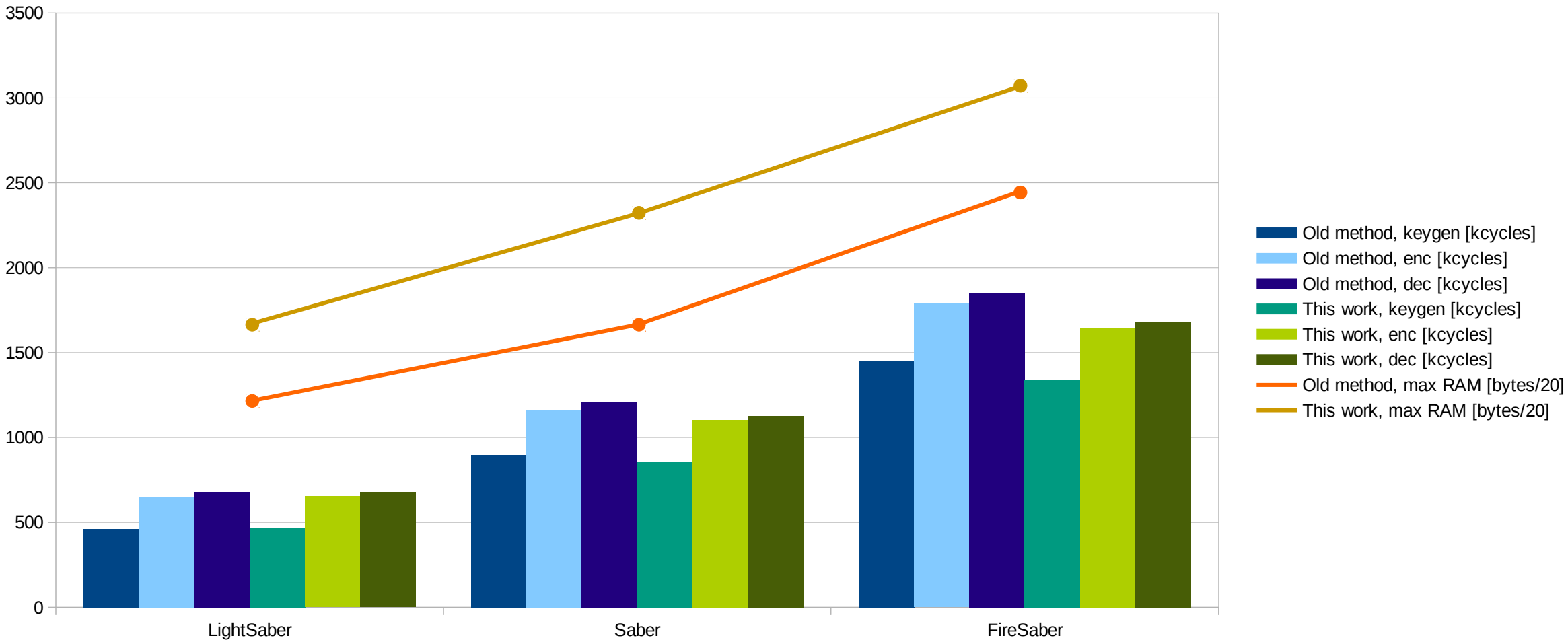
KU LEUVEN

Results – AVX2



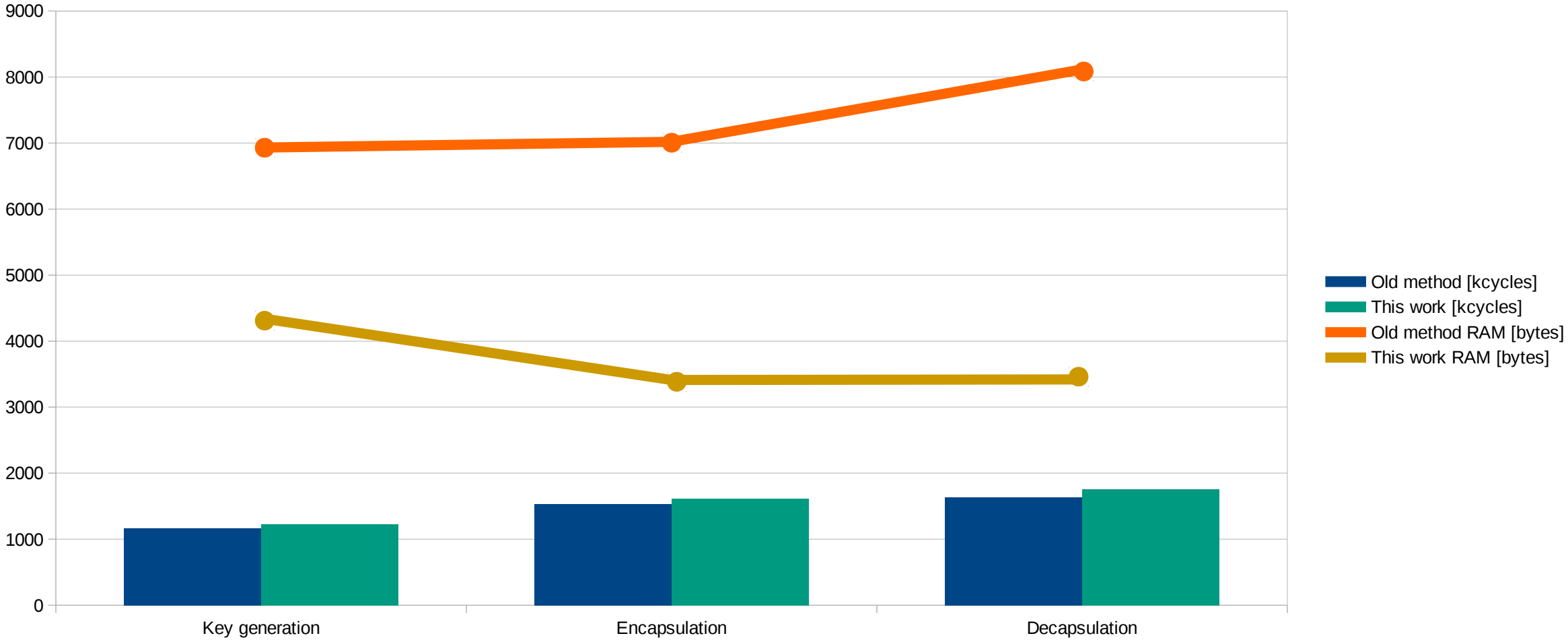
KU LEUVEN

Results – Cortex-M4 optimized for speed



KU LEUVEN

Results – Cortex-M4 Saber optimized for memory



KU LEUVEN

Conclusions

- Generalize and formalize Lazy interpolation and Pre-computation
- Show the difference between theoretical/algorithmic optimizations and real world implementations
- Fastest software implementations of Saber
- Alternatively, smallest Saber implementation for embedded platforms
- Reduced the storage required for the secret key of Saber

Thank you for your attention!