KU LEUVEN

COSIC

# Saber on ARM

## CCA-secure module lattice-based key encapsulation on ARM

Angshuman Karmakar

CHES, Amsterdam

11th September, 2018

Joint work with :

Jose Maria Bermudo Mera

Sujoy Sinha Roy

Ingrid Verbauwhede

erc European Research Council

# Saber: CCA secure post-quantum KEM*

- Module-LWR : Trade off between Standard and Ideal lattice
  - $A \in R_q^{k \times k}, \ R_q = \mathbb{Z}_q[x]/(x^{256}+1)$
  - Inherent noise→ Less randomness
- Efficient
- Flexible :
  - Increase/decrease matrix dimension to increase/decrease security
  - Basic operations stay same→ High code reusability
- All moduli are power of two
  - **Easy rounding**
  - **Easy modular reduction in HW/SW**
  - **Precludes use of NTT**

- Combination of Toom-Cook, Karatsuba and Schoolbook.

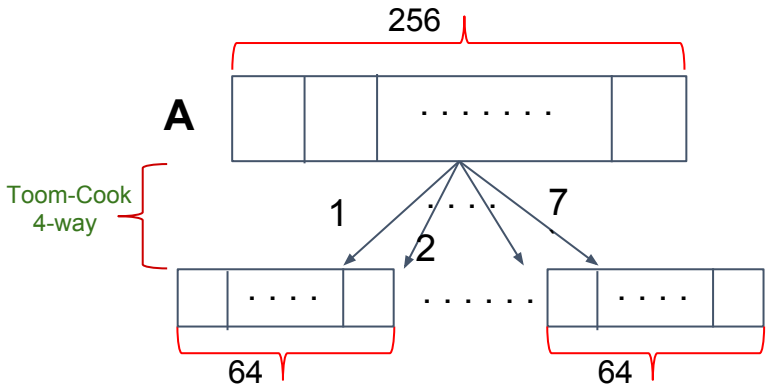
J.-P. D'Anvers, A. Karmakar, S. Sinha Roy, and F. Vercauteren. Saber: Module-lwr based key exchange, cpa-secure encryption and cca-secure kem. In A. Joux, A. Nitaj, and T. Rachidi, editors, Progress in Cryptology – AFRICACRYPT 2018, https://eprint.iacr.org/2018/230.pdf

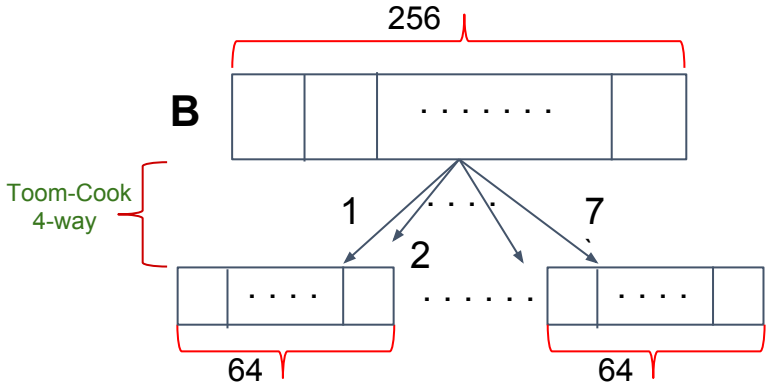
# Polynomial Multiplication

# Polynomial multiplication C=A X B

## Toom-Cook+Karatsuba+School-book

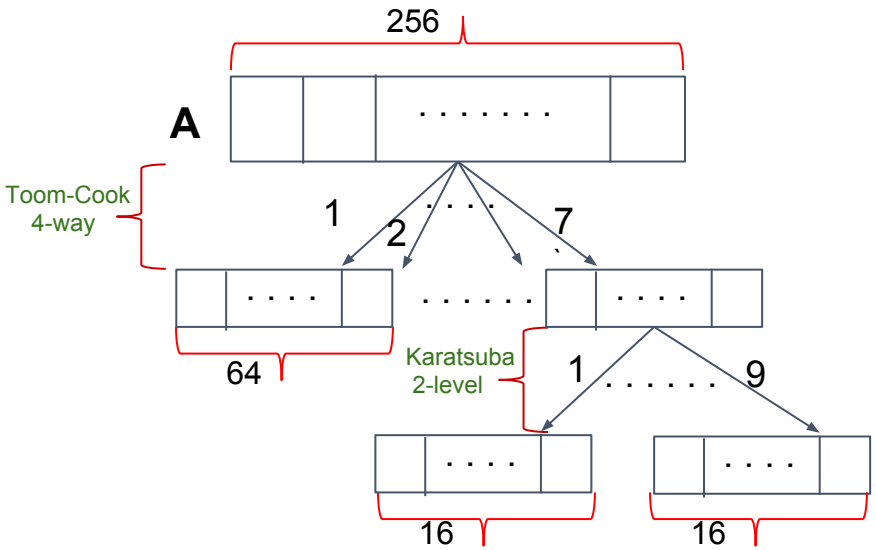
- A and B are polynomials of size 256.

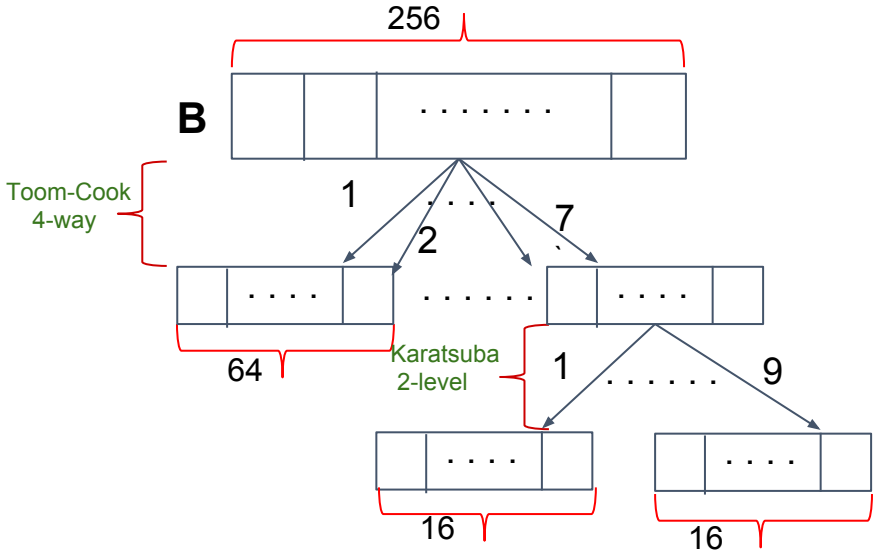


# Polynomial multiplication C=A X B

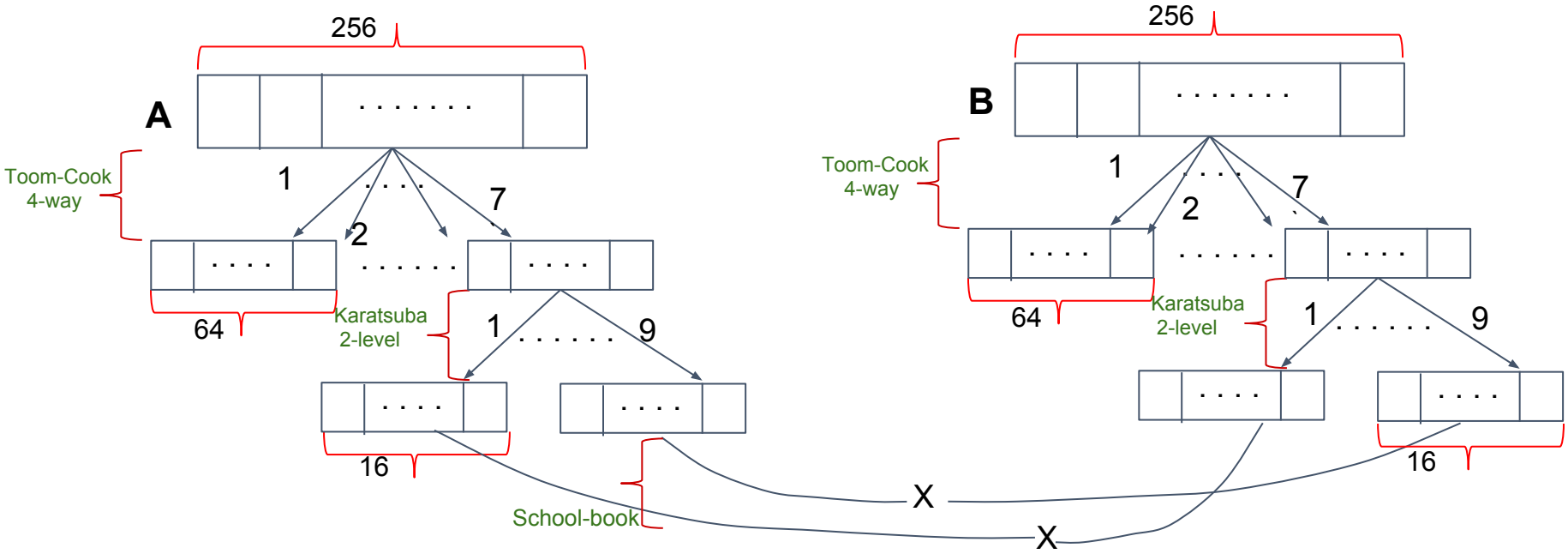
## Toom-Cook+Karatsuba+School-book



# Polynomial multiplication C=A X B

## Toom-Cook+Karatsuba+School-book



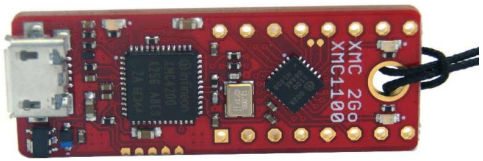
# This work

# Goal

- Saber is very efficient on high-end processors
- We show that, Saber is also efficient on low end processors like Cortex-M0 and Cortex-M4

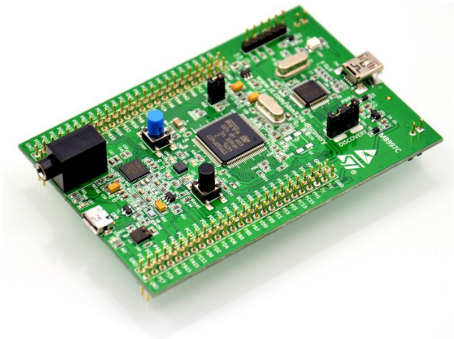
Cortex-M0: XMC2Go by Infineon

- Reduced instruction set
- Only 8 registers for data processing instructions
- 16 KB of RAM




Cortex-M4: STM32F4-discovery by STMicroelectronics

- DSP instructions
- 14 registers fully available
- 192 KB of RAM

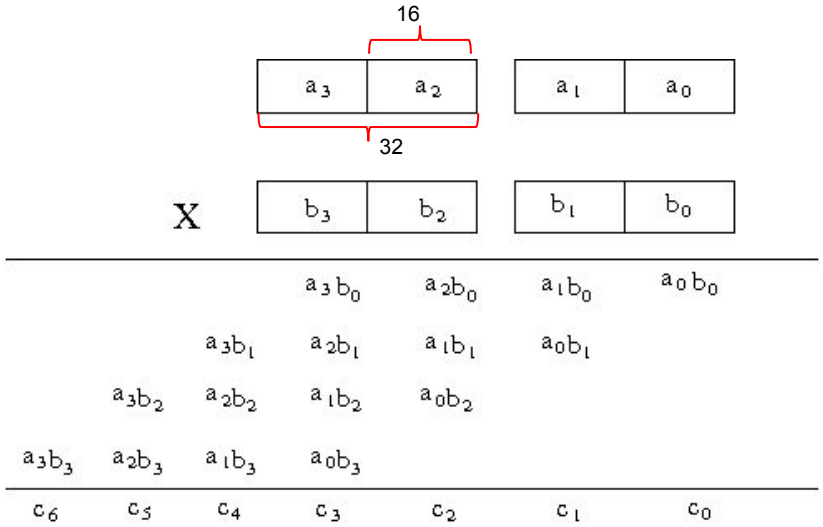


# This work

- A high speed implementation on Cortex-M4
  - Efficient use of DSP instructions on M4
    - Fewer instructions to perform a School-Book multiplication
  - An `in-register` implementation of  Toom-Cook multiplication
    - Fewer access to memory
- A memory-efficient implementation on Cortex-M0
  - A `Just-In-Time` approach to generate the elements of public matrix
  - Memory efficient in-place Karatsuba multiplication

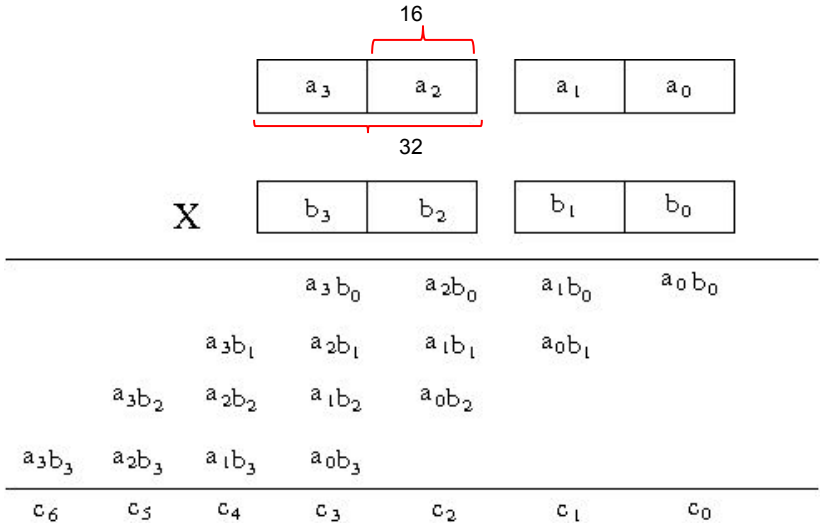
# Schoolbook multiplication

- Each coefficient is 13 bits long → fits in half word of a register
- Multiplications between half words can be done by SMLA(B/T)(B/T) instruction
  - SMLA(B/T)(B/T)($r^a$, $r^b$, $r^c$, $r^d$) := $r^a \leftarrow r^b_{0/1} * r^c_{0/1} + r^d$



# Schoolbook multiplication

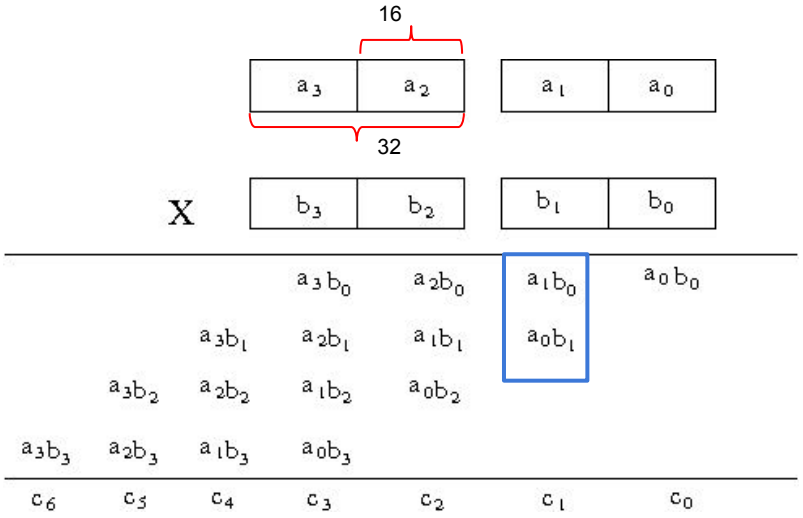
- Each coefficient fits in half word of a register
- Multiplications between half words can be done by SMLA(B/T)(B/T) instruction
  - SMLA(B/T)(B/T)($r^a$, $r^b$, $r^c$, $r^d$) := $r^a \leftarrow r^b_{0/1} * r^c_{0/1} + r^d$




16 instructions !

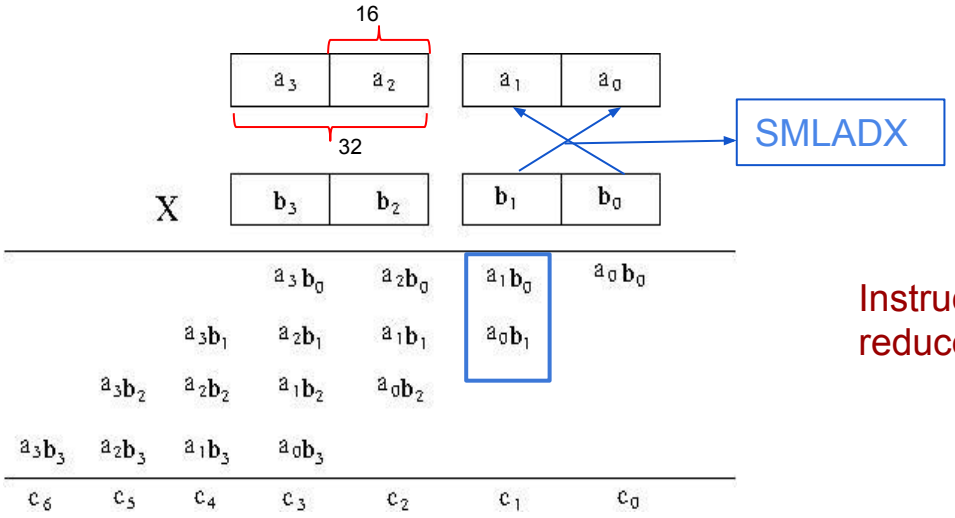
# Schoolbook multiplication

- DSP instruction SMLADX : Cross multiplies register half words
  - SMLADX($r^a$, $r^b$, $r^c$, $r^d$) := $r^a \leftarrow r^b_0 * r^c_1 + r^b_1 * r^c_0 + r^d$



# Schoolbook multiplication

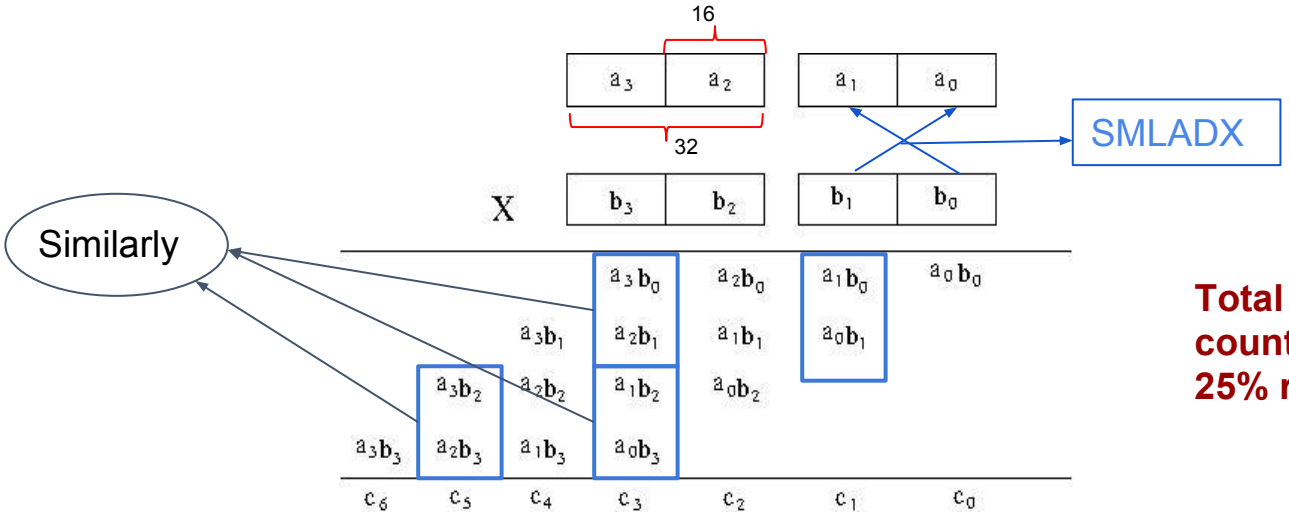
- Replace $r^a \rightarrow c_1$, $r^b \rightarrow a$, $r^c \rightarrow b$, $r^d \rightarrow 0$,
  - SMLADX($c_1$, a, b, 0) := $c_1 \leftarrow a_0 * b_1 + a_1 * b_0$




Instruction count reduces 2 → 1

# Schoolbook multiplication

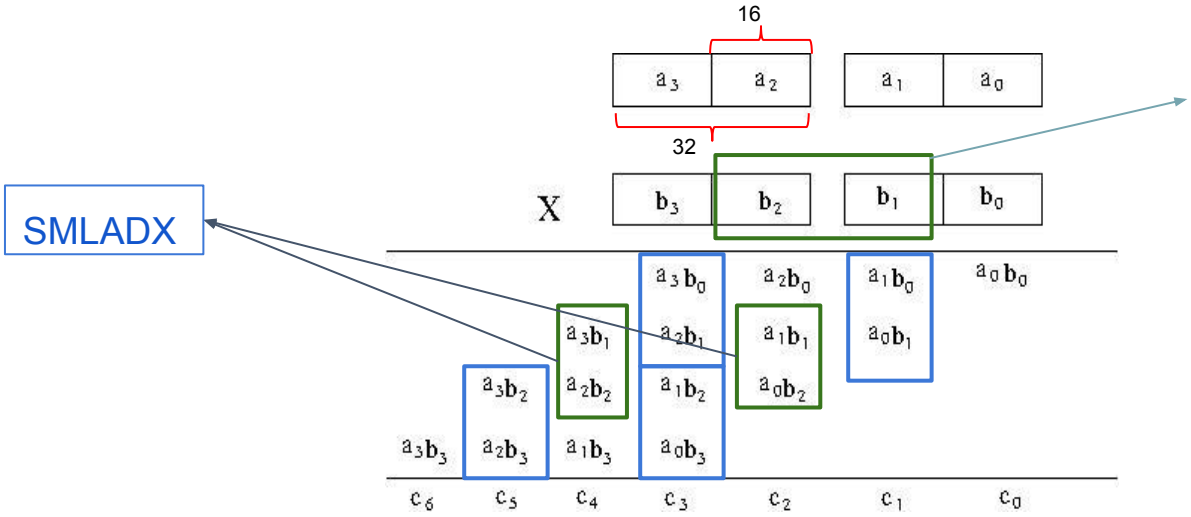
- Replace $r^a \rightarrow c_1$, $r^b \rightarrow a$, $r^c \rightarrow b$, $r^d \rightarrow 0$,
  - SMLADX($c_1$, a, b, 0) := $c_1 \leftarrow a_0 * b_1 + a_1 * b_0$




**Total Instruction count 12**
**25% reduction**

# Schoolbook multiplication

- Pack non-adjacent coefficients in spare register using PKHBT
  - Apply SMLADX again




Total Instruction count 11 !

# Schoolbook multiplication

- ≅ 37.5% reduction in instruction count for one Schoolbook multiplication
- A basic unrolled 16 X 16 multiplication requires only 168 SMLA instructions
    - A single Schoolbook multiplication takes only 587 clock cycles

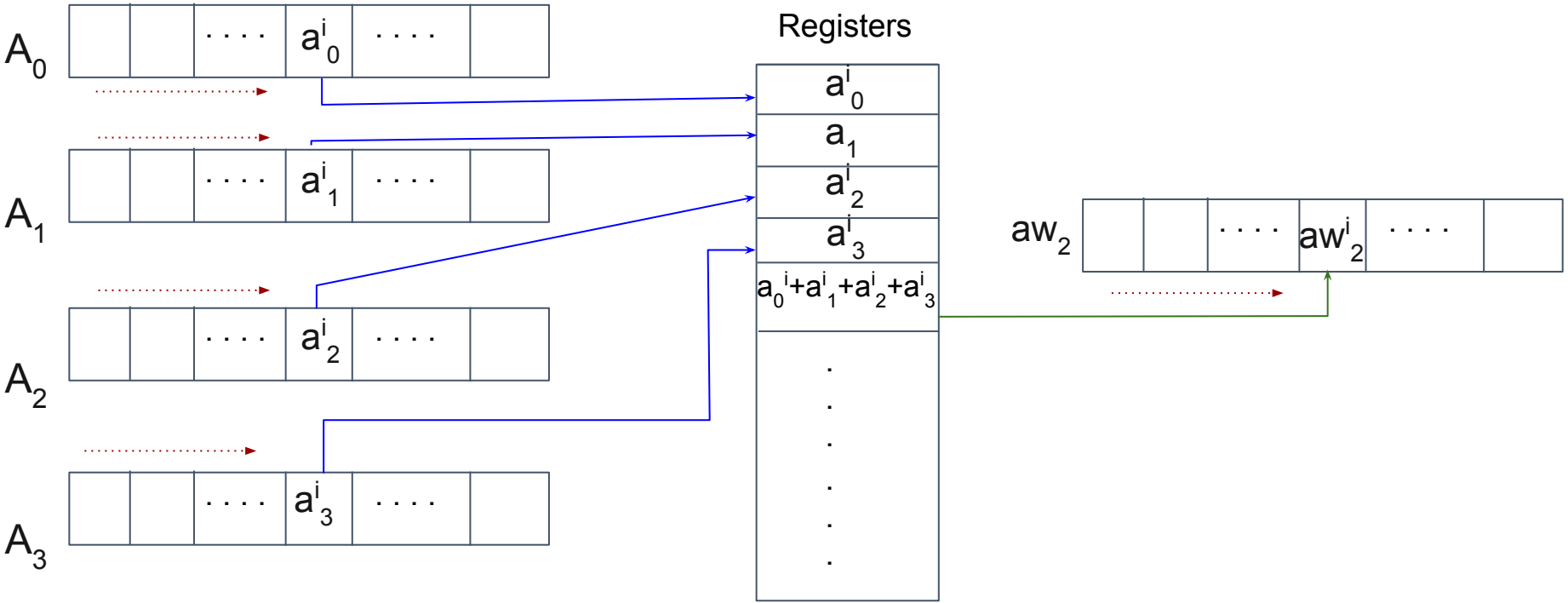
# Toom-Cook multiplication

- During evaluation phase of Toom-Cook multiplication polynomial A (& B) is divided in 4 smaller polynomials $A_3$-$A_0$ each with 64 coefficients
- Further, we need to create weighted sums of these polynomials

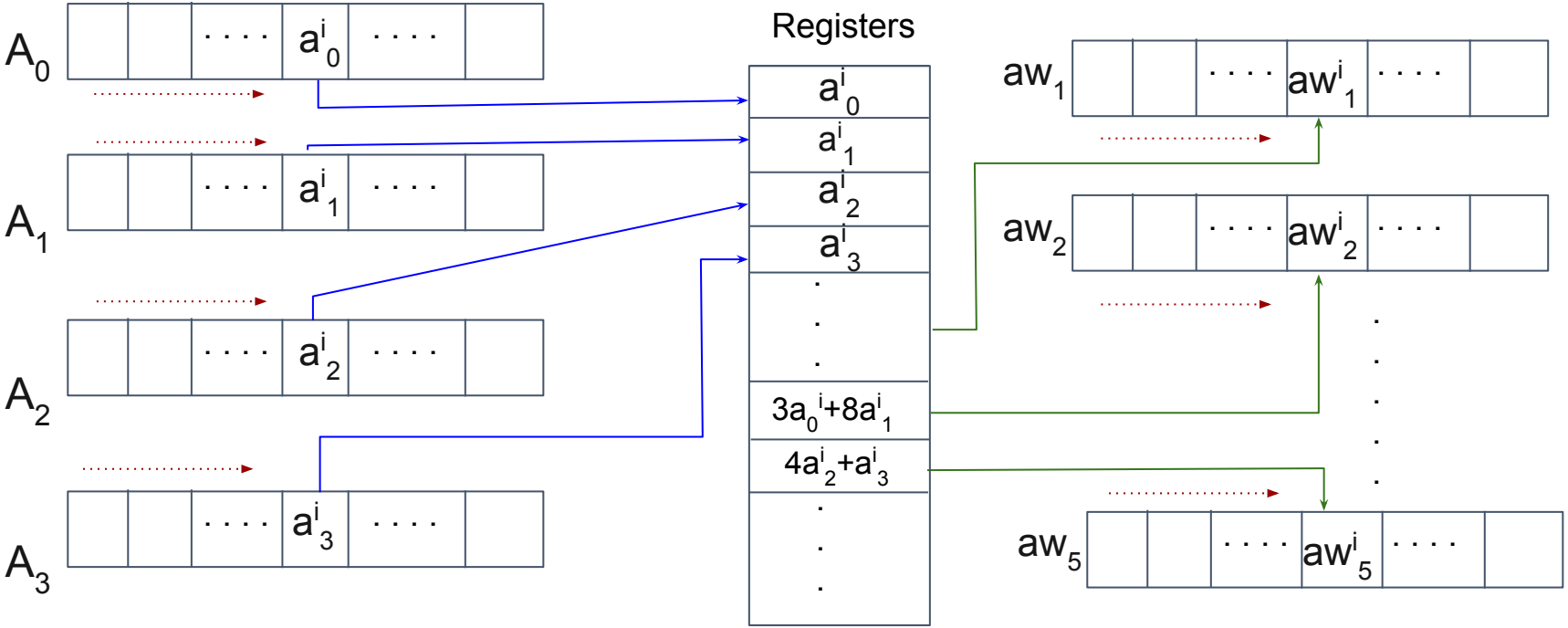
$$
\begin{aligned}
aw_1 &= A_0 + 2 \cdot A_1 + 4 \cdot A_2 + 8 \cdot A_3 \\
aw_2 &= A_0 + A_1 + A_2 + A_3 \\
&\vdots \\
aw_5 &= 8 \cdot A_0 - 4 \cdot A_1 + 2 \cdot A_2 - A_3
\end{aligned}
$$

- In a simple method, for each $aw_i$ it needs to access all 64 coefficients of $A_0$-$A_3$ *i.e* 256 memory accesses

# Normal method

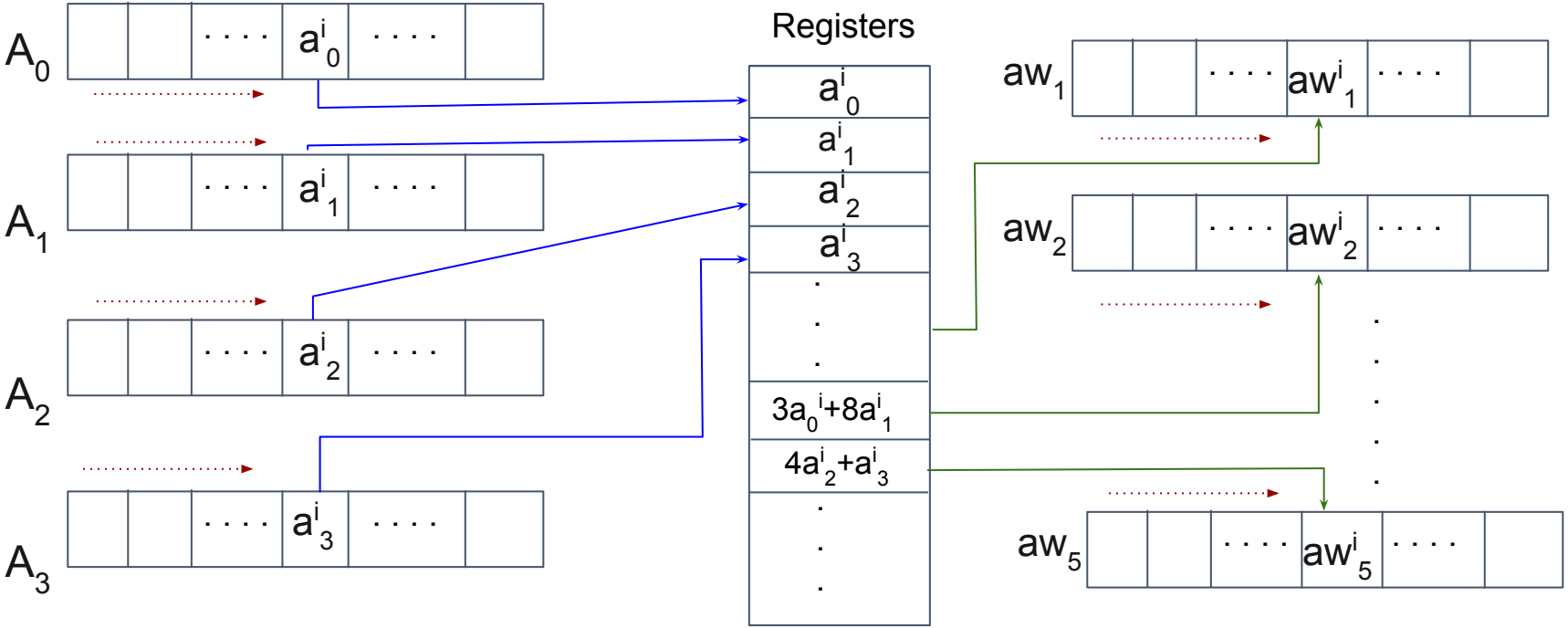


# Memory access efficient method




- *Vertical coefficient scanning* approach
- Use spare registers
- Perform more `in-register` operations to generate weighted coefficients

# Memory access efficient method

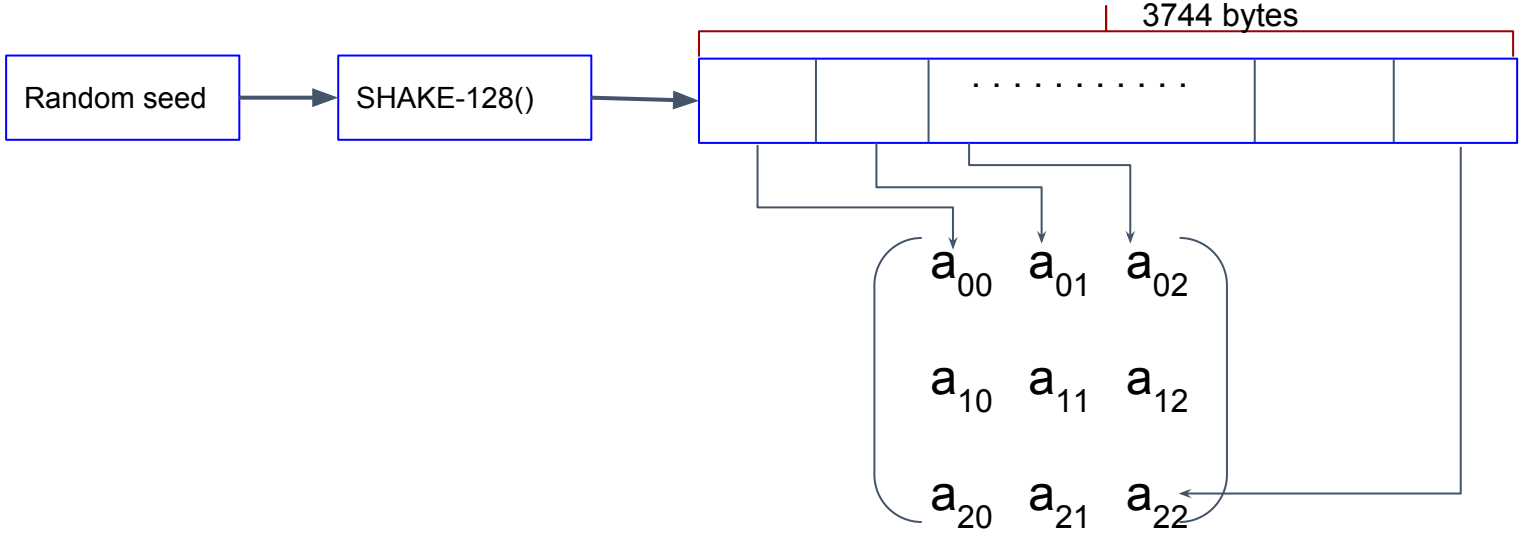



- **Number of memory accesses decrease from 5*256 to 256 only**
- Memory requirement increases.

# Memory Optimization

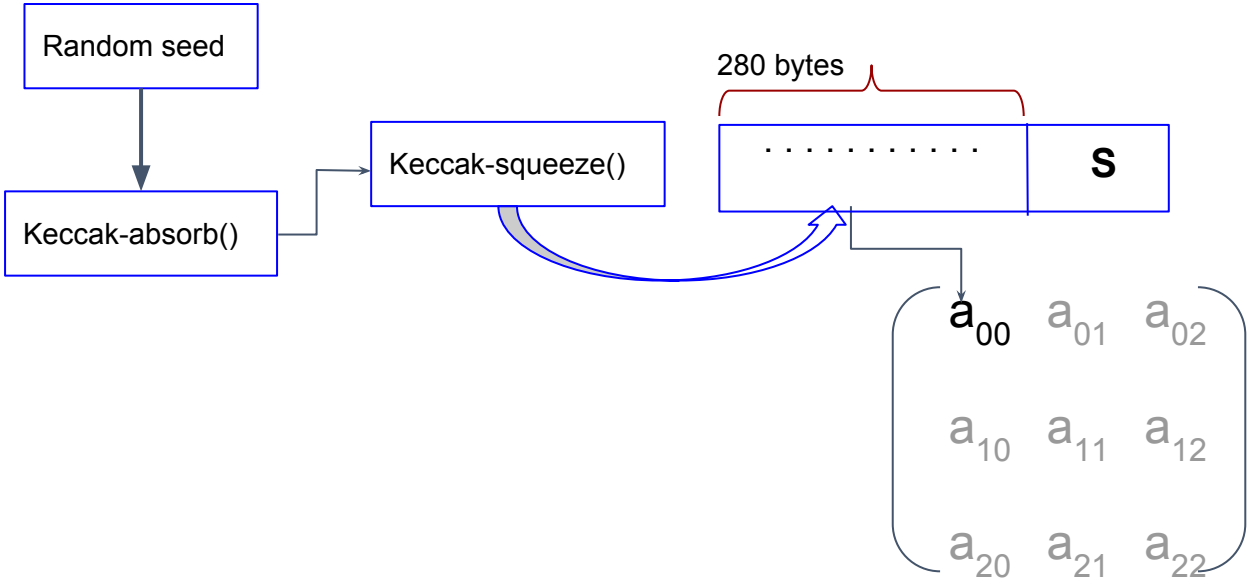
# Generation of the Public matrix.

- The public matrix $A \in R_q^{k \times k}$, requires a huge memory to generate.



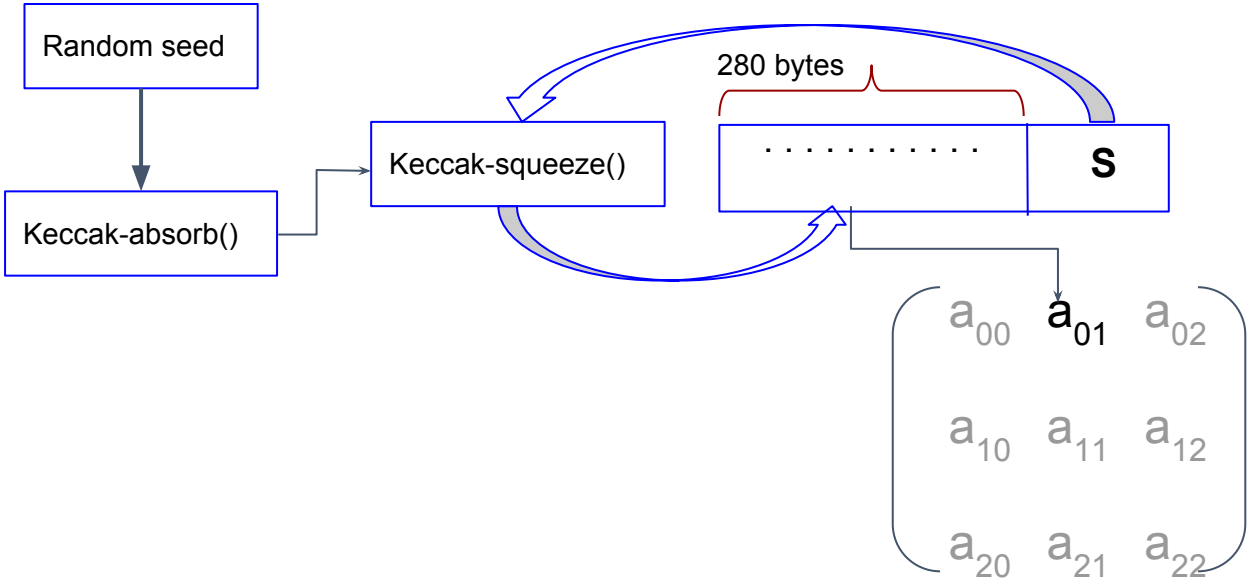
# Generation of the Public matrix.

- We use a `Just-in-Time` strategy to reuse a smaller space for each element polynomial.



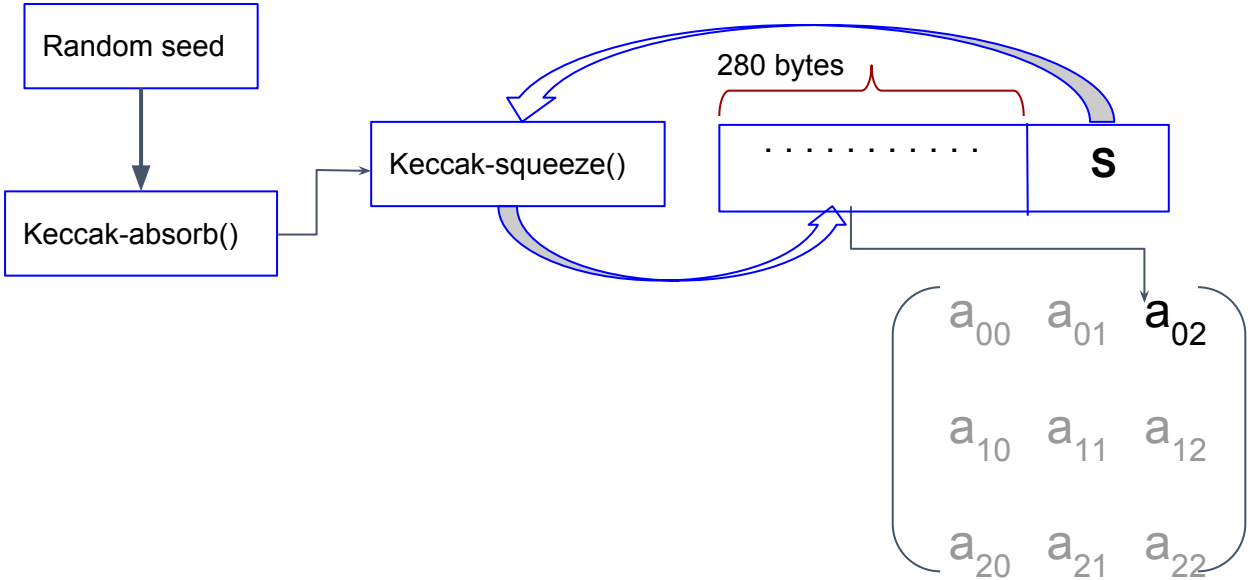
# Generation of the Public matrix.

- We use a `Just-in-Time` strategy to reuse a smaller space for each element polynomial.



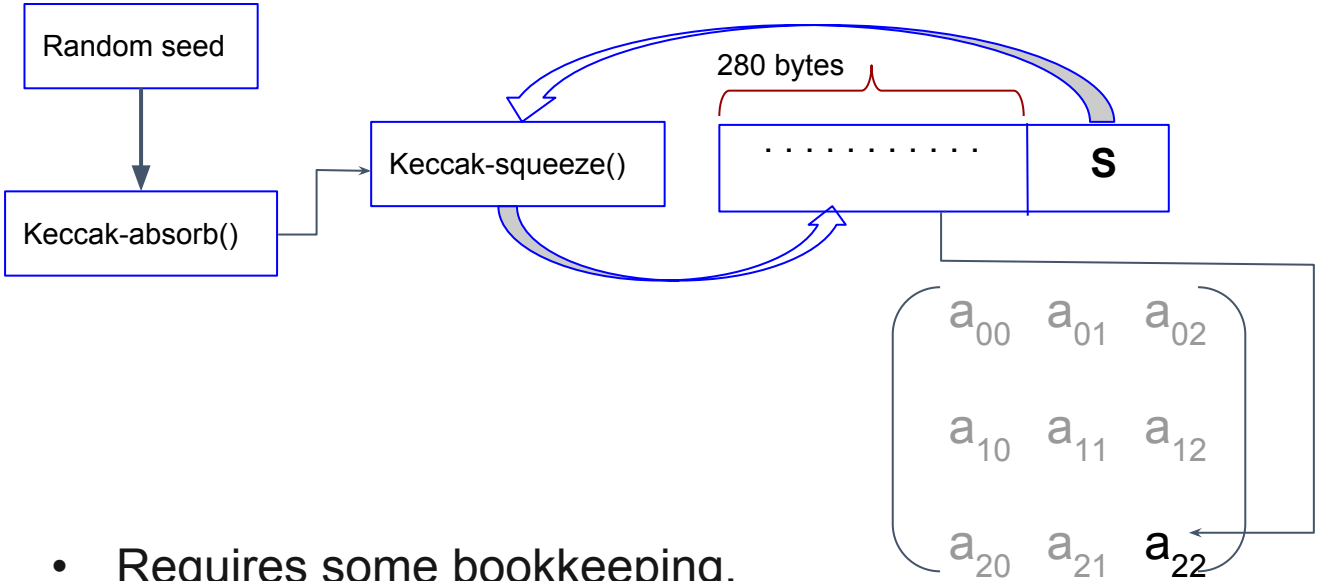
# Generation of the Public matrix.

- We use a `Just-in-Time` strategy to reuse a smaller space for each element polynomial.



# Generation of the Public matrix.

- We use a `Just-in-Time` strategy to reuse a smaller space for each element polynomial.




- Requires some bookkeeping.
- Memory requirement is ≅1/9$^{th}$ of the initial requirement

# Results
# &
# Conclusion

# Comparison to other NIST-PQC candidates

| **Cryptosystem** | **Platform** | **Key generation** [Kcycles]/[bytes] | **Encapsulation** [Kcycles]/[bytes] | **Decapsulation** [Kcycles]/[bytes] | **Multiplication** [type] |
|---|---|---|---|---|---|
| NewHope-CCA* | Cortex-M4 | 1,246 / 11,160 | 1,966 / 17,456 | 1,977 / 19,656 | NTT |
| Kyber* | Cortex-M4 | 1,200 / 10,304 | 1,497 / 13,464 | 1,526 / 14,624 | NTT |
| Saber-speed | Cortex-M4 | 1,147 / 13,883 | 1,444 / 16,667 | 1,543 / 17,763 | TC+Kara+SB |
| Saber-memory | Cortex-M4 | 1,165 / 6,931 | 1,530 / 7,019 | 1,635 / 8,115 | TC+Kara+SB |
| Saber-mem-M0 | Cortex-M0 | 4,786 / 5,031 | 6,328 / 5,119 | 7,509 / 6,215 | TC+Kara+SB |

*pqm4 post-quantum crypto library for the arm cortex-m4. https://github.com/ mupq/pqm4, 2018. [ accessed 15-April-2018]

# Conclusion

- Module-Lattice based cryptography can be practical on resource constrained devices
  - Cortex-M0 → max memory≅6.2 KB, run time < 250 ms
    - Memory requirement 1/3rd of the reference implementation
  - Cortex-M4 → max memory≅17 KB, run time < 9 ms
    - Run time 5-8 times less than the reference
- The optimizations can be applied on top of each other.
- Choice of parameters is very crucial
- For small dimensions, asymptotically slower Toom-Cook, Karatsuba multiplication can be very competitive
  - Irregular memory access of NTT
  - Utilization of special instructions

Thank you !