Meeting 9

What I did

- Skimmed the C codebase
  - It's essentially identical to the pseudo-code
  - Not very important to hardware implementation, I think.
  - Will be useful for data verification.
- Read the CRYSTALS-Kyber math paper
  - I tried but the math proofs are a bit too hard
- Read some hardware implementation of CRYSTALS-Kyber
  - Not finished

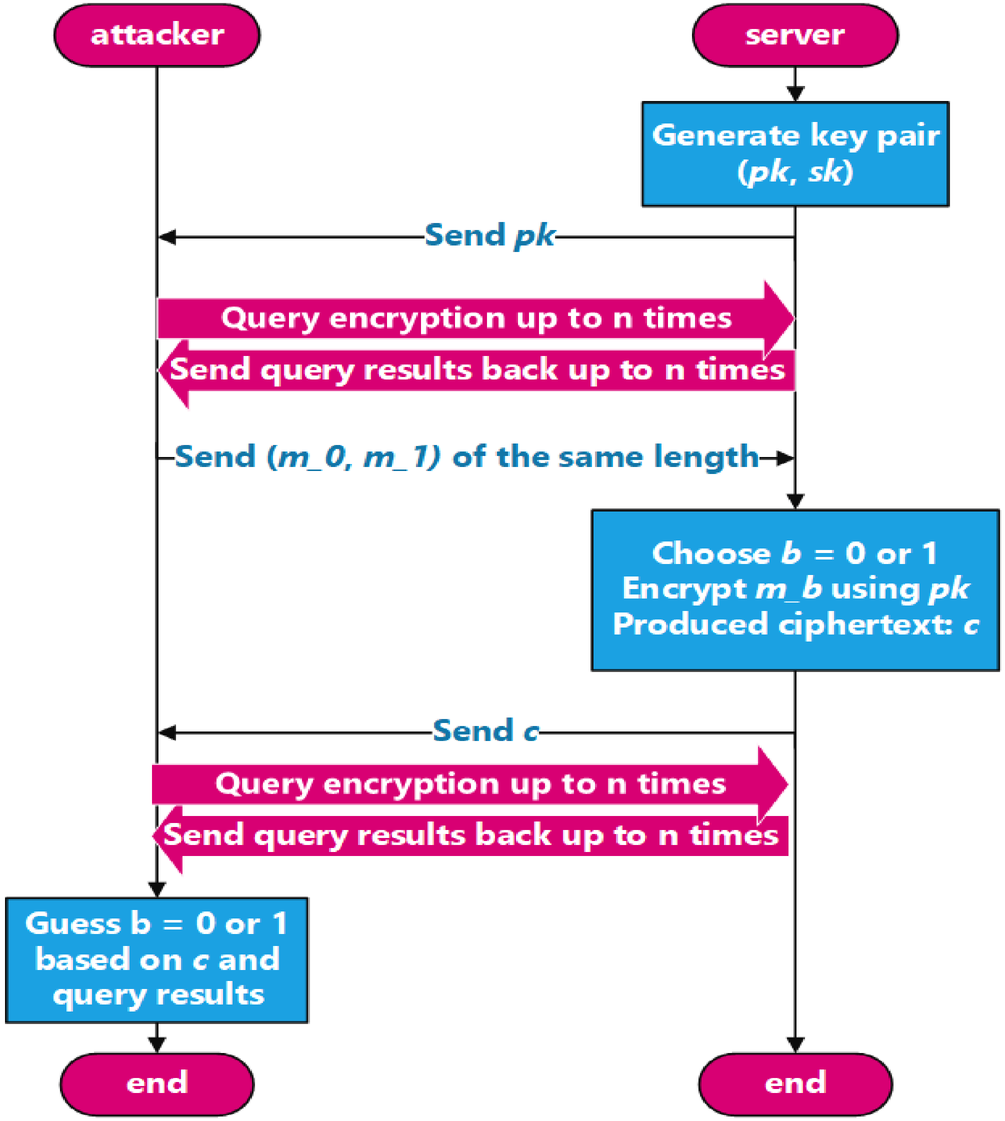
About the CRYSTALS-Kyber paper...

CRYSTALS – Kyber: a CCA-secure module-lattice-based KEM

Joppe Bos[*], Léo Ducas[†], Eike Kiltz[‡], Tancrède Lepoint[§], Vadim Lyubashevsky[¶], John M. Schanck[||], Peter Schwabe[**], Gregor Seiler[††], Damien Stehlé[‡‡],

[*]*NXP Semiconductors, Belgium. Email: joppe.bos@nxp.com*
[†]*CWI Amsterdam, The Netherlands. Email: ducas@cwi.nl*
[‡]*Ruhr-University Bochum, Germany. Email: eike.kiltz@rub.de*
[§]*SRI International, USA. Email: tancrede.lepoint@sri.com*
[¶]*IBM Research Zurich, Switzerland. Email: vad@zurich.ibm.com*
[||]*University of Waterloo, Canada. Email: jschanck@uwaterloo.ca*
[**]*Radboud University, The Netherlands. Email: peter@cryptojedi.org*
[††]*IBM Research Zurich, Switzerland. Email: grs@zurich.ibm.com*
[‡‡]*ENS de Lyon, France. Email: damien.stehle@ens-lyon.fr*

IND-CPA (In-distinguish-ability under Chosen Plaintext Attack)



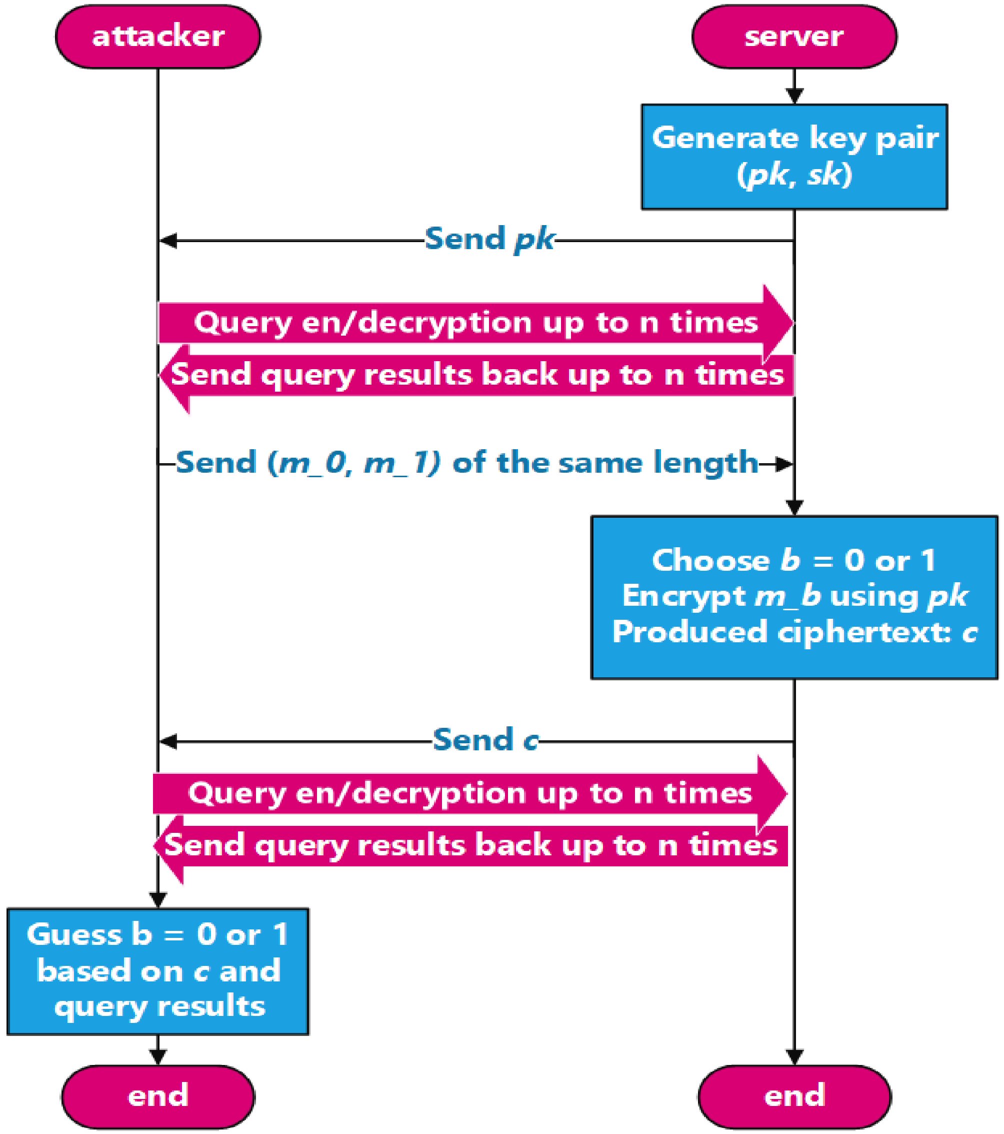
$$\mathbf{Adv}^{\text{cpa}}_{\mathsf{PKE}}(\mathsf{A})$$

$$\left| \Pr\left[b = b' \;:\; \begin{array}{l} (pk, sk) \leftarrow \mathsf{KeyGen}(); \\ (m_0, m_1, s) \leftarrow \mathsf{A}\text{[redacted]}(pk); \\ b \leftarrow \{0,1\}; c^* \leftarrow \mathsf{Enc}(pk, m_b); \\ b' \leftarrow \mathsf{A}\text{[redacted]}(s, c^*) \end{array} \right] - \frac{1}{2} \right|$$

Algorithm 2 Kyber.CPA.Enc$(pk = (\mathbf{t}, \rho), m \in \mathcal{M})$: encryption

$r \leftarrow \{0,1\}^{256}$
$\mathbf{t} := \mathsf{Decompress}_q(\mathbf{t}, d_t)$
$\mathbf{A} \sim R_q^{k \times k} := \mathsf{Sam}(\rho)$
$(\mathbf{r}, \mathbf{e}_1, e_2) \sim \beta_\eta^k \times \beta_\eta^k \times \beta_\eta := \mathsf{Sam}(r)$
$\mathbf{u} := \mathsf{Compress}_q(\mathbf{A}^T\mathbf{r} + \mathbf{e}_1, d_u)$
$v := \mathsf{Compress}_q(\mathbf{t}^T\mathbf{r} + e_2 + \lceil \frac{q}{2} \rfloor \cdot m, d_v)$
return $c := (\mathbf{u}, v)$

IND-CCA? (In-distinguish-ability under Chosen Ciphertext Attack)



$$\mathbf{Adv}^{\mathrm{cca}}_{\mathsf{PKE}}(\mathsf{A}) = \left| \Pr\left[b = b' : \begin{array}{l} (pk, sk) \leftarrow \mathsf{KeyGen}(); \\ (m_0, m_1, s) \leftarrow \mathsf{A}^{\boxed{\mathrm{DEC}(\cdot)}}(pk); \\ b \leftarrow \{0,1\}; c^* \leftarrow \mathsf{Enc}(pk, m_b); \\ b' \leftarrow \mathsf{A}^{\boxed{\mathrm{DEC}(\cdot)}}(s, c^*) \end{array} \right] - \frac{1}{2} \right|$$

Algorithm 3 Kyber.CPA.Dec($sk = \mathbf{s}, c = (\mathbf{u}, v)$): decryption

1: $\mathbf{u} := \mathsf{Decompress}_q(\mathbf{u}, d_u)$
2: $v := \mathsf{Decompress}_q(v, d_v)$
3: **return** $\mathsf{Compress}_q(v - \boxed{\mathbf{s}^T\mathbf{u}}, 1)$

secret key leaked ☹

To achieve CCA-secure → KEM (Key Encapsulation Mechanism)

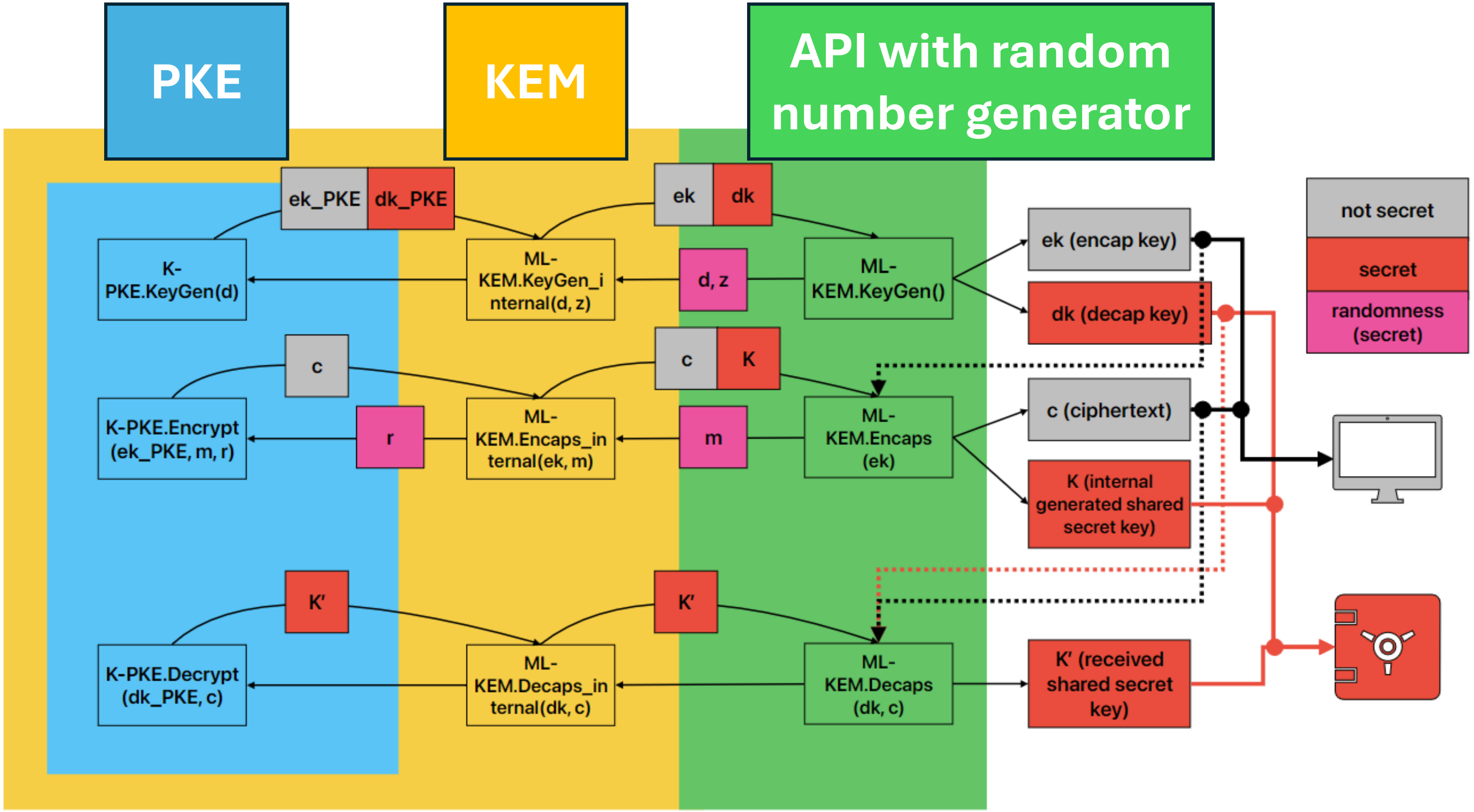
Algorithm 5 Kyber.Decaps$(sk = (\mathbf{s}, z, \mathbf{t}, \rho), c = (\mathbf{u}, v))$

1: $m' := \mathsf{Kyber.CPA.Dec}(\mathbf{s}, (\mathbf{u}, v))$
2: $(\hat{K}', r') := \mathrm{G}(\mathrm{H}(pk), m')$
3: $(\mathbf{u}', v') := \mathsf{Kyber.CPA.Enc}\left((\mathbf{t}, \rho), m'; r'\right)$
4: **if** $(\mathbf{u}', v') = (\mathbf{u}, v)$ **then**
5:     **return** $K := \mathrm{H}(\hat{K}', \mathrm{H}(c))$
6: **else**
7:     **return** $K := \mathrm{H}(z, \mathrm{H}(c))$
8: **end if**

$$\mathbf{Adv}^{\mathsf{cca}}_{\mathsf{KEM}}(\mathsf{A}) = \left| \Pr\left[b = b' \; : \; \begin{array}{l} (pk, sk) \leftarrow \mathsf{KeyGen}(); \\ b \leftarrow \{0,1\}; \\ (c^*, K_0^*) \leftarrow \mathsf{Encaps}(pk); \\ K_1^* \leftarrow \mathcal{K}; \\ b' \leftarrow \mathsf{A}^{\textsc{Decaps}(\cdot)}(pk, c^*, K_b^*) \end{array} \right] - \frac{1}{2} \right|$$

Bad ciphertext *c* that will cause <u>decryption failure</u> → re-encryption: *c'* (line 3) → Very likely: *c* ≠ *c'* Output a "not key" (line 7) → Implicitly rejected ciphertext without exposing the secret key

Thus, the layers for ML-KEM in FIPS 203



About Number Theoretic Transform (NTT)

$$R_q := \mathbb{Z}_q[X]/(X^{256}+1)$$

$$f = f_0 + f_1 X + \cdots + f_{255} X^{255} \in R_q$$

$f_1 \times f_2$: $O(256^2)$ complexity

$$X^{256}+1 = \prod_{i=0}^{127} \left(X^2 - \zeta^{2\mathsf{BitRev}_7(i)+1}\right).$$

$$T_q := \bigoplus_{i=0}^{127} \mathbb{Z}_q[X]/\left(X^2 - \zeta^{2\mathsf{BitRev}_7(i)+1}\right).$$

$\hat{g}_1 \times \hat{g}_2$: $O(256 \log 256)$ complexity → ☺

$$\hat{g} = \left(\hat{g}_{0,0} + \hat{g}_{0,1}X, \hat{g}_{1,0} + \hat{g}_{1,1}X, \ldots, \hat{g}_{127,0} + \hat{g}_{127,1}X\right) \in T_q.$$

The algorithm for NTT

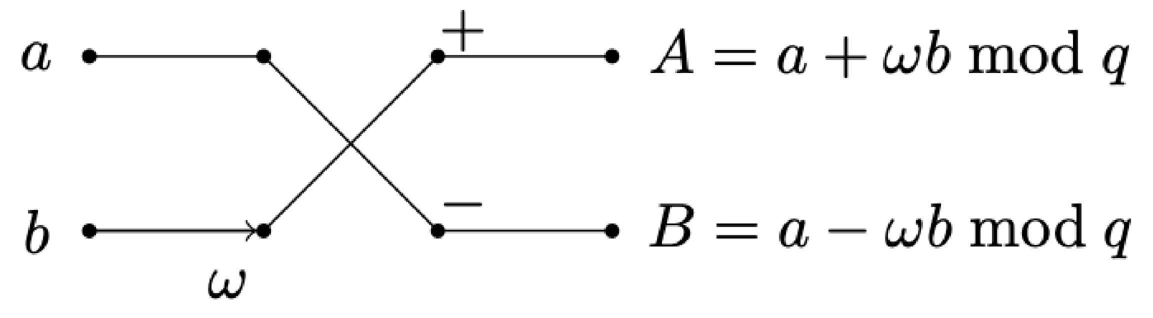
Algorithm 9 NTT(f)

Computes the NTT representation $\hat{f}$ of the given polynomial $f \in R_q$.

Input: array $f \in \mathbb{Z}_q^{256}$. ▷ the coefficients of the input polynomial

Output: array $\hat{f} \in \mathbb{Z}_q^{256}$. ▷ the coefficients of the NTT of the input polynomial

$\hat{f} \leftarrow f$ ▷ will compute in place on a copy of input array
$i \leftarrow 1$
for ($len \leftarrow 128$; $len \geq 2$; $len \leftarrow len/2$)
    for ($start \leftarrow 0$; $start < 256$; $start \leftarrow start + 2 \cdot len$)
        $zeta \leftarrow \zeta^{\mathsf{BitRev}_7(i)} \bmod q$
        $i \leftarrow i + 1$
        for ($j \leftarrow start$; $j < start + len$; j++)
            $t \leftarrow zeta \cdot \hat{f}[j + len]$
            $\hat{f}[j + len] \leftarrow \hat{f}[j] - t$
            $\hat{f}[j] \leftarrow \hat{f}[j] + t$
        end for
    end for
end for
return $\hat{f}$



(a) Cooley-Tukey butterfly

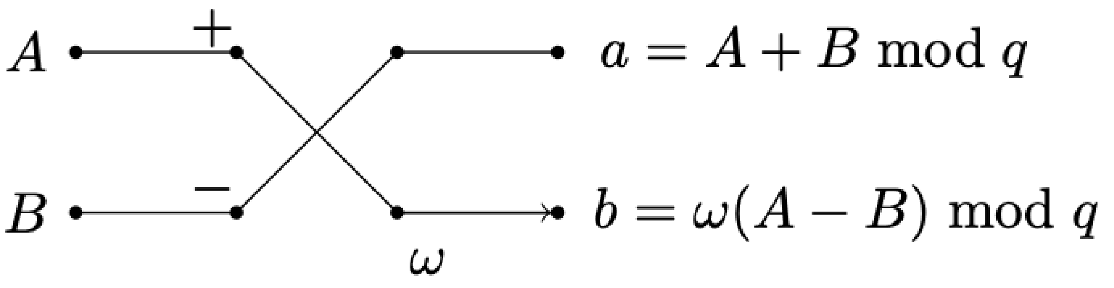
The algorithm for inverse NTT

Algorithm 10 $\mathsf{NTT}^{-1}(\hat{f})$

Computes the polynomial $f \in R_q$ that corresponds to the given NTT representation $\hat{f} \in T_q$.

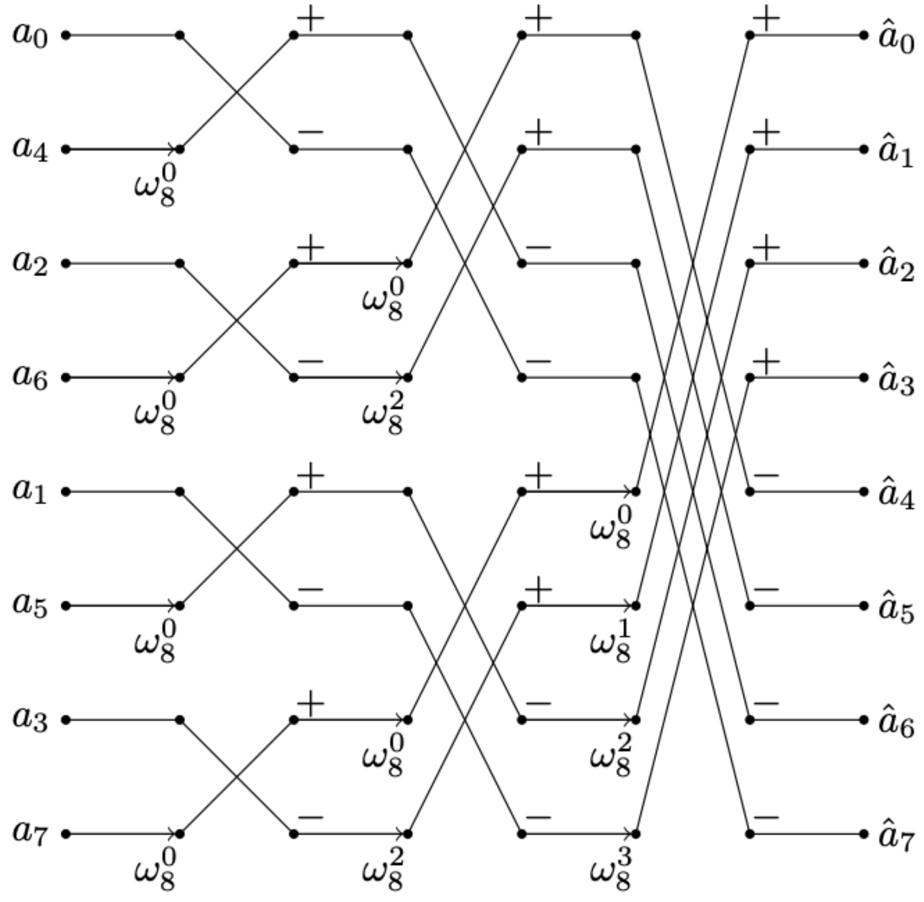
Input: array $\hat{f} \in \mathbb{Z}_q^{256}$. ▷ the coefficients of input NTT representation
Output: array $f \in \mathbb{Z}_q^{256}$. ▷ the coefficients of the inverse NTT of the input

$f \leftarrow \hat{f}$ ▷ will compute in place on a copy of input array
$i \leftarrow 127$
for ($len \leftarrow 2$; $len \leq 128$; $len \leftarrow 2 \cdot len$)
    for ($start \leftarrow 0$; $start < 256$; $start \leftarrow start + 2 \cdot len$)
        $zeta \leftarrow \zeta^{\mathsf{BitRev}_7(i)} \bmod q$
        $i \leftarrow i - 1$
        for ($j \leftarrow start$; $j < start + len$; j++)
            $t \leftarrow f[j]$
            $f[j] \leftarrow t + f[j + len]$
            $f[j + len] \leftarrow zeta \cdot (f[j + len] - t)$
        end for
    end for
end for
$f \leftarrow f \cdot 3303 \bmod q$ ▷ multiply every entry by $3303 \equiv 128^{-1} \bmod q$
return f

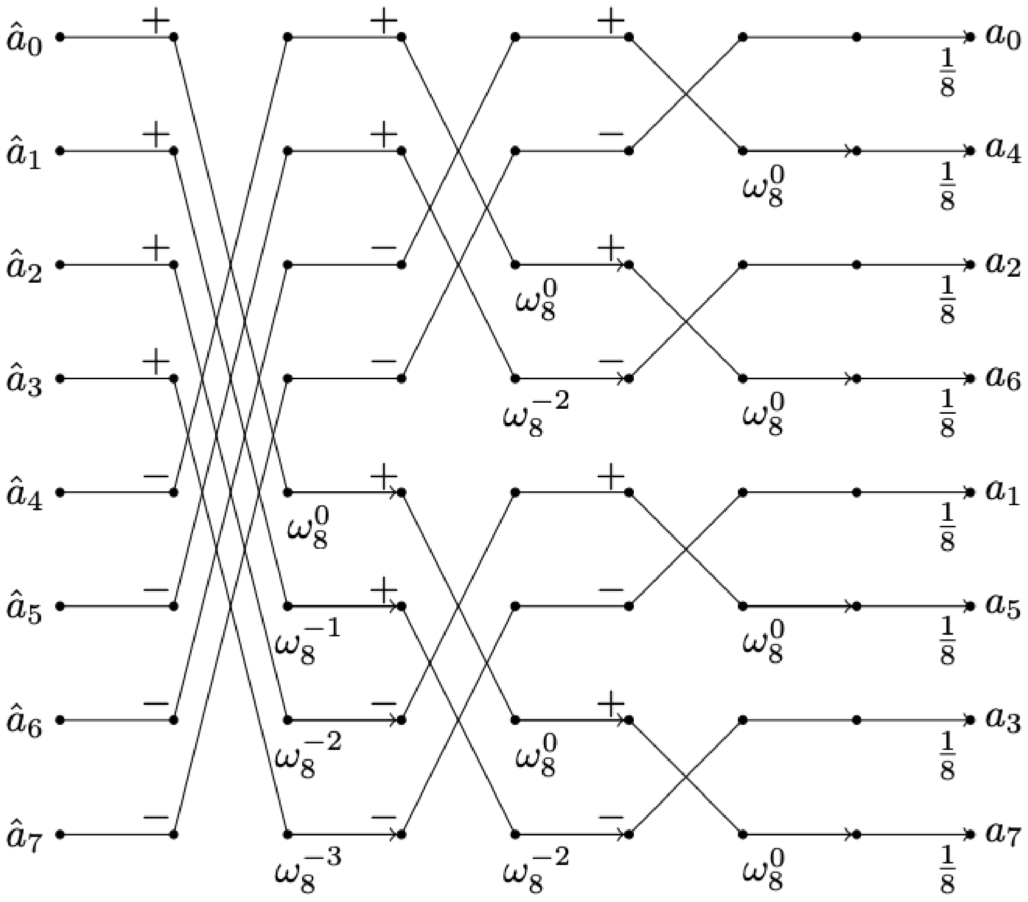


(b) Gentleman-Sande butterfly

Cooley-Tukey and Gentleman-Sande “butterfly”



(a) $\mathsf{NTT}^{CT}_{bo\to no}$



(d) $\mathsf{INTT}^{GS}_{no\to bo}$

The hardware implementation paper that I’m still reading

A Configurable CRYSTALS-Kyber Hardware Implementation with Side-Channel Protection

ARPAN JATI, NTU, Singapore and IIIT Delhi
NAINA GUPTA and ANUPAM CHATTOPADHYAY, NTU
SOMITRA KUMAR SANADHYA, IIT Jodhpur

What needs to be done next

- Hardware modules for the math operations (NTT, modulo, etc)
- Read more about past hardware implementations
- Attacks and mitigations on hardware implementations

Thanks for listening

:P