Divide and Surrender:

Exploiting Variable Division Instruction Timing in HQC Key Recovery Attacks

Robin Leander Schröder[1,2], Stefan Gast[3] and Qian Guo[4]

August 16, 2024

1. Fraunhofer SIT, Darmstadt, Germany
2. Fraunhofer Austria, Vienna, Austria
3. Graz University of Technology, Austria
4. Lund University, Sweden

Main Contributions

- Identified a division timing side-channel vulnerability in HQC KEM

Main Contributions

- Identified a division timing side-channel vulnerability in HQC KEM
- New SMT-based exploitation strategy

Main Contributions

- Identified a division timing side-channel vulnerability in HQC KEM
- New SMT-based exploitation strategy
- New key-recovery method for HQC

Main Contributions

- Identified a division timing side-channel vulnerability in HQC KEM
- New SMT-based exploitation strategy
- New key-recovery method for HQC
- Evaluation of the attack on an AMD Zen2 CPU
  - Median attack runtime less than 2 minutes

Hamming Quasi-Cyclic (HQC)

NIST Post-Quantum Key Encapsulation Mechanism (KEM) Standardization

Main selected algorithm: Kyber (lattice-based)

Round 4:

- Code Based:
  - Classic McEliece
  - Hamming Quasi-Cyclic (HQC)
  - BIKE
- SIKE (isogeny-based, broken)

HQC KEM

Key Encapsulation Mechanism (KEM) consists of three algorithms:
(KeyGen, Encaps, Decaps)

- $(\mathsf{pk}, \mathsf{sk}) \xleftarrow{\$} \mathsf{KeyGen}()$
- $(k_0, c) \xleftarrow{\$} \mathsf{Encaps}(\mathsf{pk})$
- $k_1 \leftarrow \mathsf{Decaps}(\mathsf{sk}, c)$

HQC KeyGen [Agu+22]

$\mathcal{R} := \mathbb{F}_2[X]/\langle X^n - 1\rangle$

Algorithm 1 KeyGen()

1: $(x, y) \leftarrow\$ \mathcal{R}_w \times \mathcal{R}_w$
2: sk $\leftarrow (x, y)$
3: $h \leftarrow\$ \mathcal{R}$; h corresponds to parity-check matrix $H = (I_n \ \mathrm{rot}(h))$
4: $s = x + h \cdot y$
5: pk $\leftarrow (h, s)$
6: **return** pk, sk

e.g. with parameters $n = 17669$, $w = 66$

HQC KeyGen [Agu+22]

$\mathcal{R} := \mathbb{F}_2[X]/\langle X^n - 1\rangle$

Algorithm 2 KeyGen()

1: $(x, y) \leftarrow\$ \mathcal{R}_w \times \mathcal{R}_w$
2: sk $\leftarrow (x, y)$
3: $h \leftarrow\$ \mathcal{R}$; h corresponds to parity-check matrix $H = (I_n \ \mathrm{rot}(h))$
4: $s = x + h \cdot y$
5: pk $\leftarrow (h, s)$
6: **return** pk, sk

e.g. with parameters $n = 17669$, $w = 66$

HQC KeyGen [Agu+22]

$\mathcal{R} := \mathbb{F}_2[X]/\langle X^n - 1\rangle$

Algorithm 3 KeyGen()

1: $(x, y) \leftarrow\$ \mathcal{R}_w \times \mathcal{R}_w$
2: sk $\leftarrow (x, y)$
3: $h \leftarrow\$ \mathcal{R}$; h corresponds to parity-check matrix $H = (I_n \ \mathrm{rot}(h))$
4: $s = x + h \cdot y$
5: pk $\leftarrow (h, s)$
6: **return** pk, sk

e.g. with parameters $n = 17669$, $w = 66$

HQC KeyGen [Agu+22]

$\mathcal{R} := \mathbb{F}_2[X]/\langle X^n - 1\rangle$

Algorithm 4 KeyGen()

1: $(x, y) \leftarrow\$ \mathcal{R}_w \times \mathcal{R}_w$
2: sk $\leftarrow (x, y)$
3: $h \leftarrow\$ \mathcal{R}$; h corresponds to parity-check matrix $H = (I_n \ \mathrm{rot}(h))$
4: $s = x + h \cdot y$
5: pk $\leftarrow (h, s)$
6: **return** pk, sk

e.g. with parameters $n = 17669$, $w = 66$

HQC KeyGen [Agu+22]

$\mathcal{R} := \mathbb{F}_2[X]/\langle X^n - 1\rangle$

Algorithm 5 KeyGen()

1: $(x, y) \leftarrow\$ \mathcal{R}_w \times \mathcal{R}_w$
2: sk $\leftarrow (x, y)$
3: $h \leftarrow\$ \mathcal{R}$; h corresponds to parity-check matrix $H = (I_n \ \mathrm{rot}(h))$
4: $s = x + h \cdot y$
5: pk $\leftarrow (h, s)$
6: **return** pk, sk

e.g. with parameters $n = 17669$, $w = 66$

HQC KeyGen [Agu+22]

$\mathcal{R} := \mathbb{F}_2[X]/\langle X^n - 1\rangle$

Algorithm 6 KeyGen()

1: $(x, y) \leftarrow\$ \mathcal{R}_w \times \mathcal{R}_w$
2: sk $\leftarrow (x, y)$
3: $h \leftarrow\$ \mathcal{R}$; h corresponds to parity-check matrix $H = (I_n \ \mathrm{rot}(h))$
4: $s = x + h \cdot y$
5: pk $\leftarrow (h, s)$
6: **return** pk, sk

e.g. with parameters $n = 17669$, $w = 66$

HQC PKE: Encryption and Decryption

Algorithm 7 Encrypt(pk, m)

1: $e, r_1, r_2 \xleftarrow{\$} \mathcal{R}_{\omega_e} \times \mathcal{R}_{\omega_r} \times \mathcal{R}_{\omega_r}$
2: $u \leftarrow r_1 + h \cdot r_2$
3: $v \leftarrow \mathcal{C}.\mathsf{Encode}(m) + s \cdot r_2 + e$
4: **return** $c = (u, v)$

Algorithm 8 Decrypt($\mathsf{sk} = (x, y), c = (u, v)$)

1: **return** $\mathcal{C}.\mathsf{Decode}(v - u \cdot y)$

HQC PKE: Encryption and Decryption

Algorithm 9 Encrypt(pk, m)

1: $e, r_1, r_2 \xleftarrow{\$} \mathcal{R}_{\omega_e} \times \mathcal{R}_{\omega_r} \times \mathcal{R}_{\omega_r}$
2: $u \leftarrow r_1 + h \cdot r_2$
3: $v \leftarrow \mathcal{C}.\text{Encode}(m) + s \cdot r_2 + e$
4: **return** $c = (u, v)$

Algorithm 10 Decrypt(sk $= (x, y)$, $c = (u, v)$)

1: **return** $\mathcal{C}.\text{Decode}(v - u \cdot y)$

HQC PKE: Encryption and Decryption

Algorithm 11 Encrypt(pk, m)

1: $e, r_1, r_2 \xleftarrow{\$} \mathcal{R}_{\omega_e} \times \mathcal{R}_{\omega_r} \times \mathcal{R}_{\omega_r}$
2: $u \leftarrow r_1 + h \cdot r_2$
3: $v \leftarrow \mathcal{C}.\mathsf{Encode}(m) + s \cdot r_2 + e$
4: **return** $c = (u, v)$

Algorithm 12 Decrypt(sk $= (x, y)$, $c = (u, v)$)

1: **return** $\mathcal{C}.\mathsf{Decode}(v - u \cdot y)$

HQC PKE: Encryption and Decryption

Algorithm 13 Encrypt(pk, m)

1: $e, r_1, r_2 \xleftarrow{\$} \mathcal{R}_{\omega_e} \times \mathcal{R}_{\omega_r} \times \mathcal{R}_{\omega_r}$
2: $u \leftarrow r_1 + h \cdot r_2$
3: $v \leftarrow \mathcal{C}.\mathsf{Encode}(m) + s \cdot r_2 + e$
4: **return** $c = (u, v)$

Algorithm 14 Decrypt(sk $= (x, y)$, $c = (u, v)$)

1: **return** $\mathcal{C}.\mathsf{Decode}(v - u \cdot y)$

HQC PKE: Encryption and Decryption

Algorithm 15 Encrypt(pk, m)

1: $e, r_1, r_2 \xleftarrow{\$} \mathcal{R}_{\omega_e} \times \mathcal{R}_{\omega_r} \times \mathcal{R}_{\omega_r}$
2: $u \leftarrow r_1 + h \cdot r_2$
3: $v \leftarrow \mathcal{C}.\text{Encode}(m) + s \cdot r_2 + e$
4: **return** $c = (u, v)$

Algorithm 16 Decrypt(sk $= (x, y)$, $c = (u, v)$)

1: **return** $\mathcal{C}.\text{Decode}(v - u \cdot y)$

HQC PKE: Encryption and Decryption

Algorithm 17 Encrypt(pk, m)

1: $e, r_1, r_2 \xleftarrow{\$} \mathcal{R}_{\omega_e} \times \mathcal{R}_{\omega_r} \times \mathcal{R}_{\omega_r}$
2: $u \leftarrow r_1 + h \cdot r_2$
3: $v \leftarrow \mathcal{C}.\mathsf{Encode}(m) + s \cdot r_2 + e$
4: **return** $c = (u, v)$

Algorithm 18 Decrypt(sk $= (x, y)$, $c = (u, v)$)

1: **return** $\mathcal{C}.\mathsf{Decode}(v - u \cdot y)$

Decryption Failures

Decoding is successful when $v - u \cdot y$ has $\leq \delta$ errors:

$$
\begin{aligned}
& v - u \cdot y \\
=& \mathcal{C}.\mathsf{Encode}(m) + s \cdot r_2 + e - (r_1 + h \cdot r_2) \cdot y \\
=& \mathcal{C}.\mathsf{Encode}(m) + (x + h \cdot y) \cdot r_2 + e - (r_1 + h \cdot r_2) \cdot y \\
=& \mathcal{C}.\mathsf{Encode}(m) + \underbrace{x \cdot r_2 + e - r_1 \cdot y}_{\text{sparse}}
\end{aligned}
$$

HQC KEM: Encaps and Decaps

Algorithm 19 KEM.Encaps

Input: pk

Output: K, $c = (u, v)$, salt

1: $m \leftarrow\$ \mathbb{F}_2^k$
2: $\text{salt} \leftarrow\$ \mathbb{F}_2^{128}$
3: $\theta \leftarrow \mathcal{G}(m \| \text{pk} \| \text{salt})$
4: $c \leftarrow$PKE.Encrypt(pk, m, θ)
5: $K \leftarrow \mathcal{K}(m, c)$

Algorithm 20 KEM.Decaps

Input: $\text{sk} = (x, y)$, $c = (u, v)$, salt

Output: K

1: $m' \leftarrow$ PKE.Decrypt(sk, c)
2: $\theta' \leftarrow \mathcal{G}(m' \| \text{pk} \| \text{salt})$
3: $c' \leftarrow$ PKE.Encrypt(pk, m', θ')[a]
4: **if** $m' = \perp \ \vee \ c \neq c'$ **then**
5: $\quad K \leftarrow \mathcal{K}(\sigma, c)$
6: **else**
7: $\quad K \leftarrow \mathcal{K}(m', c)$
8: **end if**

HQC KEM: Encaps and Decaps

Algorithm 21 KEM.Encaps

Input: pk

Output: K, $c = (u, v)$, salt

1: $m \leftarrow\$ \mathbb{F}_2^k$
2: $\text{salt} \leftarrow\$ \mathbb{F}_2^{128}$
3: $\theta \leftarrow \mathcal{G}(m \| \text{pk} \| \text{salt})$
4: $c \leftarrow$ PKE.Encrypt(pk, m, θ)
5: $K \leftarrow \mathcal{K}(m, c)$

Algorithm 22 KEM.Decaps

Input: $\text{sk} = (x, y)$, $c = (u, v)$, salt

Output: K

1: $m' \leftarrow$ PKE.Decrypt(sk, c)
2: $\theta' \leftarrow \mathcal{G}(m' \| \text{pk} \| \text{salt})$
3: $c' \leftarrow$ PKE.Encrypt(pk, m', θ')[a]
4: **if** $m' = \perp \ \vee \ c \neq c'$ **then**
5: $\quad K \leftarrow \mathcal{K}(\sigma, c)$
6: **else**
7: $\quad K \leftarrow \mathcal{K}(m', c)$
8: **end if**

HQC KEM: Encaps and Decaps

Algorithm 23 KEM.Encaps

Input: pk

Output: K, $c = (u, v)$, salt

1: $m \leftarrow\$ \mathbb{F}_2^k$
2: $\text{salt} \leftarrow\$ \mathbb{F}_2^{128}$
3: $\theta \leftarrow \mathcal{G}(m \| \text{pk} \| \text{salt})$
4: $c \leftarrow$PKE.Encrypt(pk, m, θ)
5: $K \leftarrow \mathcal{K}(m, c)$

Algorithm 24 KEM.Decaps

Input: $\text{sk} = (x, y)$, $c = (u, v)$, salt

Output: K

1: $m' \leftarrow$ PKE.Decrypt(sk, c)
2: $\theta' \leftarrow \mathcal{G}(m' \| \text{pk} \| \text{salt})$
3: $c' \leftarrow$ PKE.Encrypt(pk, m', θ')[a]
4: **if** $m' = \perp \ \vee \ c \neq c'$ **then**
5: $\quad K \leftarrow \mathcal{K}(\sigma, c)$
6: **else**
7: $\quad K \leftarrow \mathcal{K}(m', c)$
8: **end if**

HQC KEM: Encaps and Decaps

Algorithm 25 KEM.Encaps

Input: pk

Output: K, $c = (u, v)$, salt

1: $m \leftarrow\$ \mathbb{F}_2^k$
2: $\text{salt} \leftarrow\$ \mathbb{F}_2^{128}$
3: $\theta \leftarrow \mathcal{G}(m \| \text{pk} \| \text{salt})$
4: $c \leftarrow$PKE.Encrypt(pk, m, θ)
5: $K \leftarrow \mathcal{K}(m, c)$

Algorithm 26 KEM.Decaps

Input: $\text{sk} = (x, y)$, $c = (u, v)$, salt

Output: K

1: $m' \leftarrow$ PKE.Decrypt(sk, c)
2: $\theta' \leftarrow \mathcal{G}(m' \| \text{pk} \| \text{salt})$
3: $c' \leftarrow$ PKE.Encrypt(pk, m', θ')[a]
4: **if** $m' = \perp \ \vee \ c \neq c'$ **then**
5: $\quad K \leftarrow \mathcal{K}(\sigma, c)$
6: **else**
7: $\quad K \leftarrow \mathcal{K}(m', c)$
8: **end if**

HQC KEM: Encaps and Decaps

Algorithm 27 KEM.Encaps

Input: pk

Output: K, $c = (u, v)$, salt

1: $m \leftarrow\$ \mathbb{F}_2^k$
2: $\text{salt} \leftarrow\$ \mathbb{F}_2^{128}$
3: $\theta \leftarrow \mathcal{G}(m \| \text{pk} \| \text{salt})$
4: $c \leftarrow$PKE.Encrypt(pk, m, θ)
5: $K \leftarrow \mathcal{K}(m, c)$

Algorithm 28 KEM.Decaps

Input: $\text{sk} = (x, y)$, $c = (u, v)$, salt

Output: K

1: $m' \leftarrow$ PKE.Decrypt(sk, c)
2: $\theta' \leftarrow \mathcal{G}(m' \| \text{pk} \| \text{salt})$
3: $c' \leftarrow$ PKE.Encrypt(pk, m', θ')[a]
4: **if** $m' = \perp \ \vee \ c \neq c'$ **then**
5: $\quad K \leftarrow \mathcal{K}(\sigma, c)$
6: **else**
7: $\quad K \leftarrow \mathcal{K}(m', c)$
8: **end if**

HQC KEM: Encaps and Decaps

Algorithm 29 KEM.Encaps

Input: pk

Output: K, $c = (u, v)$, salt

1: $m \leftarrow\$ \mathbb{F}_2^k$
2: $\text{salt} \leftarrow\$ \mathbb{F}_2^{128}$
3: $\theta \leftarrow \mathcal{G}(m \| \text{pk} \| \text{salt})$
4: $c \leftarrow$PKE.Encrypt(pk, m, θ)
5: $K \leftarrow \mathcal{K}(m, c)$

Algorithm 30 KEM.Decaps

Input: $\text{sk} = (x, y)$, $c = (u, v)$, salt

Output: K

1: $m' \leftarrow$ PKE.Decrypt(sk, c)
2: $\theta' \leftarrow \mathcal{G}(m' \| \text{pk} \| \text{salt})$
3: $c' \leftarrow$ PKE.Encrypt(pk, m', θ')[a]
4: **if** $m' = \perp \ \vee \ c \neq c'$ **then**
5: $\quad K \leftarrow \mathcal{K}(\sigma, c)$
6: **else**
7: $\quad K \leftarrow \mathcal{K}(m', c)$
8: **end if**

HQC KEM: Encaps and Decaps

Algorithm 31 KEM.Encaps

Input: pk

Output: K, $c = (u, v)$, salt

1: $m \leftarrow\$ \mathbb{F}_2^k$
2: $\text{salt} \leftarrow\$ \mathbb{F}_2^{128}$
3: $\theta \leftarrow \mathcal{G}(m \| \text{pk} \| \text{salt})$
4: $c \leftarrow$PKE.Encrypt(pk, m, θ)
5: $K \leftarrow \mathcal{K}(m, c)$

Algorithm 32 KEM.Decaps

Input: $\text{sk} = (x, y)$, $c = (u, v)$, salt

Output: K

1: $m' \leftarrow$ PKE.Decrypt(sk, c)
2: $\theta' \leftarrow \mathcal{G}(m' \| \text{pk} \| \text{salt})$
3: $c' \leftarrow$ PKE.Encrypt(pk, m', θ')[a]
4: **if** $m' = \perp \ \vee \ c \neq c'$ **then**
5: $\quad K \leftarrow \mathcal{K}(\sigma, c)$
6: **else**
7: $\quad K \leftarrow \mathcal{K}(m', c)$
8: **end if**

HQC KEM: Encaps and Decaps

Algorithm 33 KEM.Encaps

Input: pk

Output: K, $c = (u, v)$, salt

1: $m \leftarrow\$ \mathbb{F}_2^k$
2: $\text{salt} \leftarrow\$ \mathbb{F}_2^{128}$
3: $\theta \leftarrow \mathcal{G}(m \| \text{pk} \| \text{salt})$
4: $c \leftarrow$PKE.Encrypt(pk, m, θ)
5: $K \leftarrow \mathcal{K}(m, c)$

Algorithm 34 KEM.Decaps

Input: $\text{sk} = (x, y)$, $c = (u, v)$, salt

Output: K

1: $m' \leftarrow$ PKE.Decrypt(sk, c)
2: $\theta' \leftarrow \mathcal{G}(m' \| \text{pk} \| \text{salt})$
3: $c' \leftarrow$ PKE.Encrypt(pk, m', θ')[a]
4: **if** $m' = \perp \ \vee \ c \neq c'$ **then**
5: $\quad K \leftarrow \mathcal{K}(\sigma, c)$
6: **else**
7: $\quad K \leftarrow \mathcal{K}(m', c)$
8: **end if**

HQC KEM: Encaps and Decaps

Algorithm 35 KEM.Encaps

Input: pk

Output: K, $c = (u, v)$, salt

1: $m \leftarrow\$ \mathbb{F}_2^k$
2: $\text{salt} \leftarrow\$ \mathbb{F}_2^{128}$
3: $\theta \leftarrow \mathcal{G}(m \| \text{pk} \| \text{salt})$
4: $c \leftarrow$PKE.Encrypt(pk, m, θ)
5: $K \leftarrow \mathcal{K}(m, c)$

Algorithm 36 KEM.Decaps

Input: $\text{sk} = (x, y)$, $c = (u, v)$, salt

Output: K

1: $m' \leftarrow$ PKE.Decrypt(sk, c)
2: $\theta' \leftarrow \mathcal{G}(m' \| \text{pk} \| \text{salt})$
3: $c' \leftarrow$ PKE.Encrypt(pk, m', θ')[a]
4: **if** $m' = \perp \ \vee \ c \neq c'$ **then**
5: $\quad K \leftarrow \mathcal{K}(\sigma, c)$
6: **else**
7: $\quad K \leftarrow \mathcal{K}(m', c)$
8: **end if**

[a] [Guo+22]

HQC KEM: Encaps and Decaps

Algorithm 37 KEM.Encaps

Input: pk

Output: K, $c = (u, v)$, salt

1: $m \leftarrow\$ \mathbb{F}_2^k$
2: $\text{salt} \leftarrow\$ \mathbb{F}_2^{128}$
3: $\theta \leftarrow \mathcal{G}(m \| \text{pk} \| \text{salt})$
4: $c \leftarrow$PKE.Encrypt(pk, m, θ)
5: $K \leftarrow \mathcal{K}(m, c)$

Algorithm 38 KEM.Decaps

Input: $\text{sk} = (x, y)$, $c = (u, v)$, salt

Output: K

1: $m' \leftarrow$ PKE.Decrypt(sk, c)
2: $\theta' \leftarrow \mathcal{G}(m' \| \text{pk} \| \text{salt})$
3: $c' \leftarrow$ PKE.Encrypt(pk, m', θ')[a]
4: **if** $m' = \perp \ \vee \ c \neq c'$ **then**
5: $\quad K \leftarrow \mathcal{K}(\sigma, c)$
6: **else**
7: $\quad K \leftarrow \mathcal{K}(m', c)$
8: **end if**

[a] [Guo+22]

Vulnerability & Side-Channel

Timing Vulnerability

Fix for rejection-sampling timing-attack [Guo+22]:

Constant-time algorithm [Sen21].

But: uses modulo-reductions.

```
for (size_t i = 0; i < weight; ++i) {
    tmp[i] = i + rand_u32[i] % (PARAM_N - i);
}
```

Figure 1: Vulnerable code snippet generating division (`div`) instructions.

Numerator Dependent Division Throughput

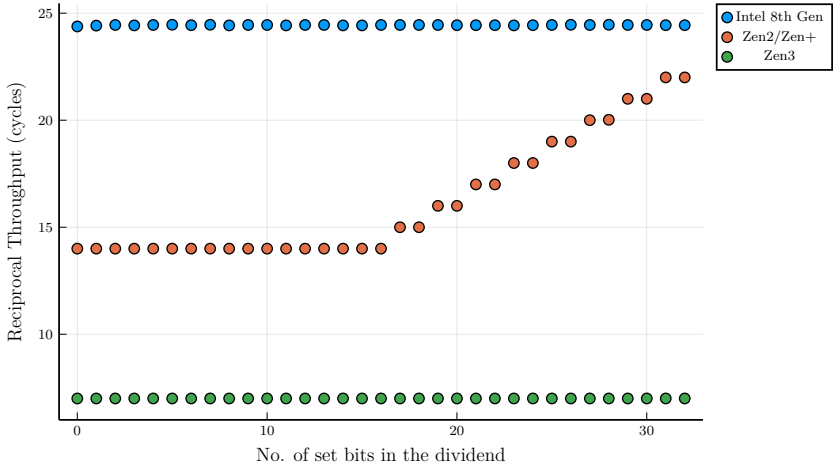
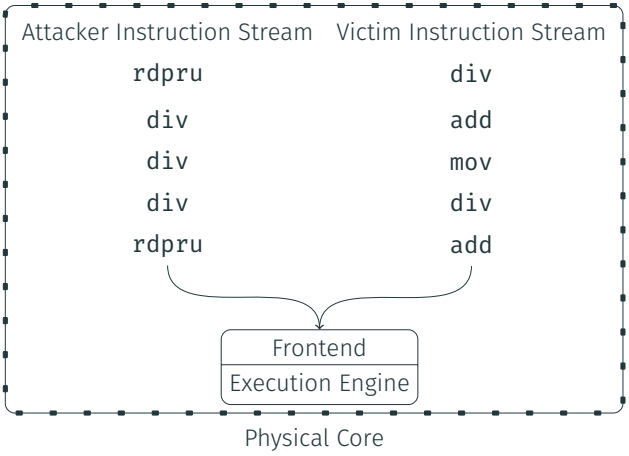
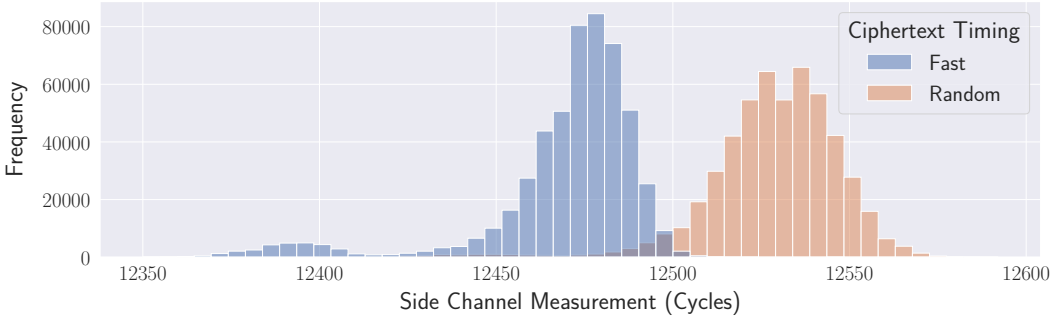


Figure 2: Division throughput for varying numerands and a fixed divisor (17669).

DIV-SMT: Simultaneous Multithreading/Hyperthreading to the rescue



Timing Difference



Exploitation

Recall: HQC encryption/decryption

Algorithm 39 Decrypt($\mathsf{sk} = (x, y), c = (u, v)$)

1: **return** $\mathcal{C}.\mathsf{Decode}(\underbrace{v - u \cdot y}_{=\mathcal{C}.\mathsf{Encode}(m)+x\cdot r_2+e-r_1\cdot y})$

Set r_1 to $\mathbf{1}$ and r_2 and e to 0: the error becomes secret key!

Recover the error of the ciphertext to recover the secret key.

Recall: HQC encryption/decryption

Algorithm 40 Decrypt($\mathsf{sk} = (x, y), c = (u, v)$)

1: **return** $\mathcal{C}.\mathsf{Decode}(\underbrace{v - u \cdot y}_{=\mathcal{C}.\mathsf{Encode}(m)+y})$

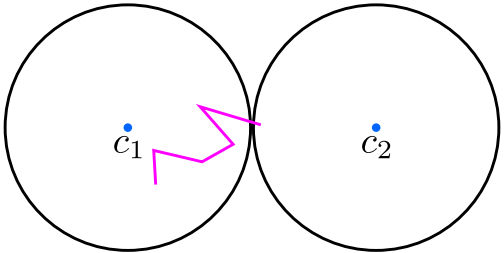
Set r_1 to $\mathbf{1}$ and r_2 and e to 0: the error becomes secret key!

Recover the error of the ciphertext to recover the secret key.

Reaction Attack [HGS99]

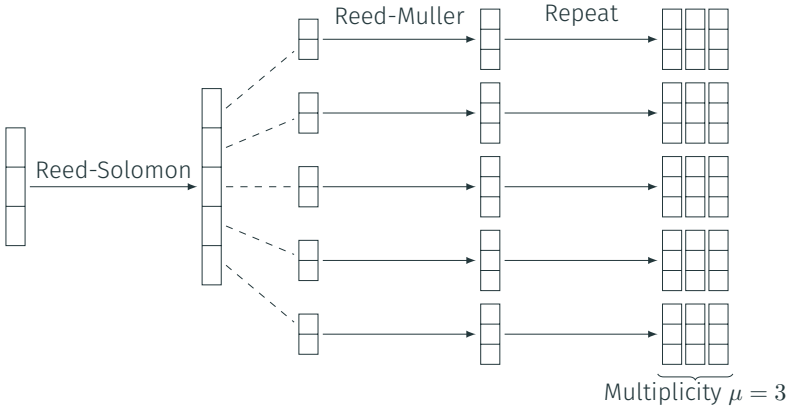
Offline: find “fast” message

Then recover the error y:

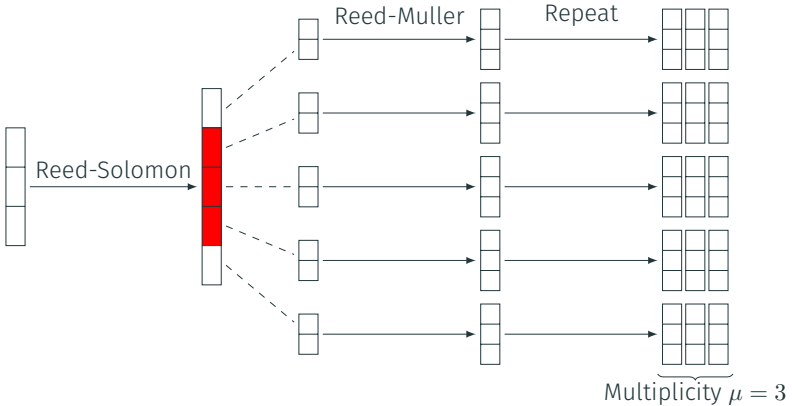


Attack Optimization

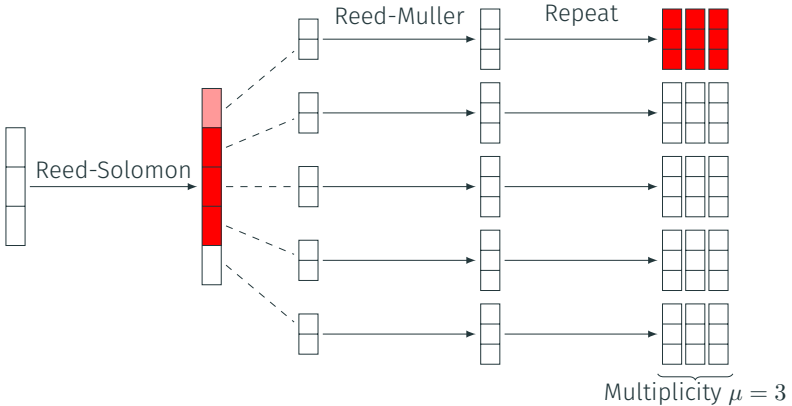
Concatenated Code



Concatenated Code



Concatenated Code



Zero Tester + Shifting

- Test if a block of 384 consecutive bits contains only zero-bits
  - Add errors that “almost” reach the decoding boundary

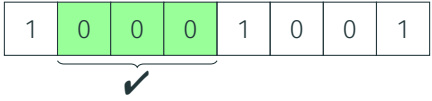
Zero Tester + Shifting

- Test if a block of 384 consecutive bits contains only zero-bits
  - Add errors that “almost” reach the decoding boundary
- Shift secret-key to zero-test arbitrary 384-bit blocks

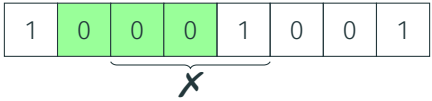
Recovering Zero Bits

Recovering Zero Bits

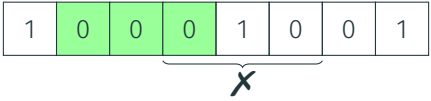
Recovering Zero Bits



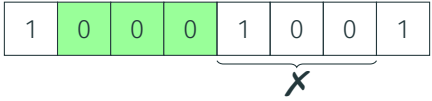
Recovering Zero Bits



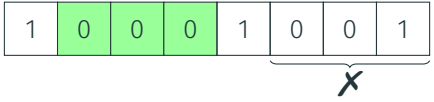
Recovering Zero Bits



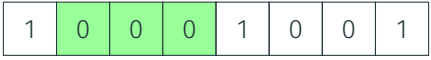
Recovering Zero Bits



Recovering Zero Bits

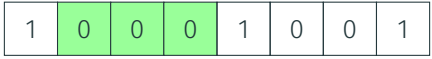


Recovering Zero Bits



How to choose good shifts?

Recovering Zero Bits



How to get entire secret key?

Recovering the Full Key

$$x + h \cdot y = s \tag{1}$$

Recover $\approx n$ out of $2n$ zero-bits with zero-tests

Recover the remaining bits using linear-algebra, integrating public-key information

Simulation Results

Work	Oracle Calls
GHJLNS22 [Guo+22]	866 143
SHRWS22 [Sch+22]	>50 000
HSCGJ23 [Hua+23]	>50 000
SCA-LDPC[Guo+23]	10 000
OurWork	1 142

	$\mathcal{O}_{HQC}^{perfect}$	$\mathcal{O}_{HQC}^{iedal}$	$\mathcal{O}_{HQC}^{0.995}$	$\mathcal{O}_{HQC}^{0.95}$	$\mathcal{O}_{HQC}^{0.9}$	$\mathcal{O}_{HQC}^{0.515}$
SCA-LDPC[Guo+23]	9 000	10 000	18 000	35 250	59 500	n/a
Our Work	1 094	1 142	2 246	4 922	6 951	5 728 728
Success Rate	78.00%	77.30%	76.90%	77.10%	76.60%	77.13%

Simulation Results

Work	Oracle Calls
GHJLNS22 [Guo+22]	866 143
SHRWS22 [Sch+22]	>50 000
HSCGJ23 [Hua+23]	>50 000
SCA-LDPC[Guo+23]	10 000
Our Work	1 142

	$\mathcal{O}_{HQC}^{perfect}$	$\mathcal{O}_{HQC}^{iedal}$	$\mathcal{O}_{HQC}^{0.995}$	$\mathcal{O}_{HQC}^{0.95}$	$\mathcal{O}_{HQC}^{0.9}$	$\mathcal{O}_{HQC}^{0.515}$
SCA-LDPC[Guo+23]	9 000	10 000	18 000	35 250	59 500	n/a
Our Work	1 094	1 142	2 246	4 922	6 951	5 728 728
Success Rate	78.00%	77.30%	76.90%	77.10%	76.60%	77.13%

Countermeasures

- Manually implement Barrett reduction
- Use a different method to map to the required range
  In BIKE: $i + \frac{(n-i)s_i}{2^{32}}$

Divide and Surrender:

Exploiting Variable Division Instruction Timing in HQC Key Recovery Attacks

Robin Leander Schröder[1,2], Stefan Gast[3] and Qian Guo[4]

August 16, 2024

1. Fraunhofer SIT, Darmstadt, Germany
2. Fraunhofer Austria, Vienna, Austria
3. Graz University of Technology, Austria
4. Lund University, Sweden

Main Contributions

- Identified a division timing side-channel vulnerability in HQC KEM
- New SMT-based exploitation strategy
- New key-recovery method for HQC
- Evaluation of the attack on an AMD Zen2 CPU
  - Median attack runtime less than 2 minutes

References

[Agu+22] Carlos Aguilar Melchor, Nicolas Aragon, Slim Bettaieb, Loïc Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, Edoardo Persichetti, Gilles Zémor, Jurjen Bos, Arnaud Dion, Jerome Lacan, Jean-Marc Robert, and Pascal Veron. ***HQC.*** Tech. rep. available at `https://csrc.nist.gov/Projects/post-quantum-cryptography/round-4-submissions`. National Institute of Standards and Technology, 2022.

[GJN20] Qian Guo, Thomas Johansson, and Alexander Nilsson. ***A key-recovery timing attack on post-quantum primitives using the Fujisaki-Okamoto transformation and its application on FrodoKEM.*** Cryptology ePrint Archive, Report 2020/743. `https://eprint.iacr.org/2020/743`. 2020.

References ii

[Guo+22] Qian Guo, Clemens Hlauschek, Thomas Johansson, Norman Lahr, Alexander Nilsson, and Robin Leander Schröder. **“Don’t Reject This: Key-Recovery Timing Attacks Due to Rejection-Sampling in HQC and BIKE”.** In: *IACR TCHES* 2022.3 (2022), pp. 223–263. DOI: 10.46586/tches.v2022.i3.223-263.

[Guo+23] Qian Guo, Denis Nabokov, Alexander Nilsson, and Thomas Johansson. **“SCA-LDPC: A Code-Based Framework for Key-Recovery Side-Channel Attacks on Post-Quantum Encryption Schemes”.** In: *ASIACRYPT 2023* (2023). https://eprint.iacr.org/2023/294. URL: https://eprint.iacr.org/2023/294.

[HGS99] Chris Hall, Ian Goldberg, and Bruce Schneier. **“Reaction attacks against several public-key cryptosystem”.** In: *Information and Communication Security: Second International Conference, ICICS’99, Sydney, Australia, November 9-11, 1999. Proceedings 2*. Springer. 1999, pp. 2–12.

References iii

[Hua+23] Senyang Huang, Rui Qi Sim, Chitchanok Chuengsatiansup, Qian Guo, and Thomas Johansson. **“Cache-Timing Attack Against HQC”.** In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2023, Issue 3 (2023). https://tches.iacr.org/index.php/TCHES/article/view/10959, pp. 136–163. DOI: 10.46586/tches.v2023.i3.136-163.

[PT19] Thales Bandiera Paiva and Routo Terada. **“A Timing Attack on the HQC Encryption Scheme”.** In: *SAC 2019*. Ed. by Kenneth G. Paterson and Douglas Stebila. Vol. 11959. LNCS. Springer, Heidelberg, 2019-08, pp. 551–573. DOI: 10.1007/978-3-030-38471-5_22.

References iv

[Sch+22] Thomas Schamberger, Lukas Holzbaur, Julian Renner, Antonia Wachter-Zeh, and Georg Sigl. **"A Power Side-Channel Attack on the Reed-Muller Reed-Solomon Version of the HQC Cryptosystem".** In: *Post-Quantum Cryptography - 13th International Workshop, PQCrypto 2022, Virtual Event, September 28-30, 2022, Proceedings*. Ed. by Jung Hee Cheon and Thomas Johansson. Vol. 13512. Lecture Notes in Computer Science. Springer, 2022, pp. 327–352. DOI: 10.1007/978-3-031-17234-2_16. URL: https://doi.org/10.1007/978-3-031-17234-2_16.

[Sen21] Nicolas Sendrier. ***Secure Sampling of Constant-Weight Words – Application to BIKE.*** Cryptology ePrint Archive, Paper 2021/1631. https://eprint.iacr.org/2021/1631. 2021. URL: https://eprint.iacr.org/2021/1631.

[WT+19] Guillaume Wafo-Tapa, Slim Bettaieb, Loic Bidoux, Philippe Gaborit, and Etienne Marcatel. ***A Practicable Timing Attack Against HQC and its Countermeasure.*** Cryptology ePrint Archive, Report 2019/909. `https://eprint.iacr.org/2019/909`. 2019.

Data Flow and Relevance

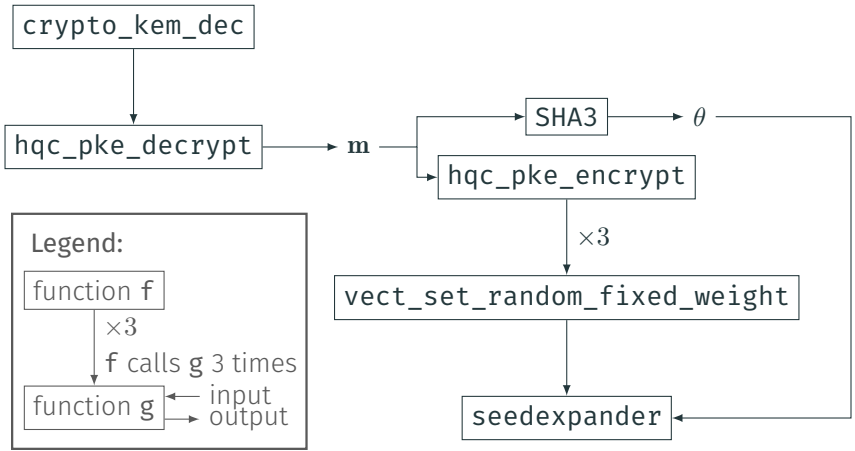


Figure 3: Data flow in Hamming Quasi-Cyclic (HQC).

HQC Decapsulation

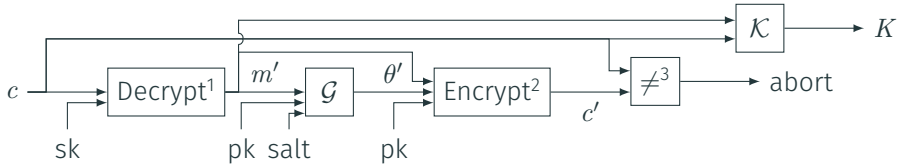


Figure 4: Decapsulation function

[1] WT+19; PT19.
[2] Guo+22.
[3] GJN20.

HQC Decapsulation

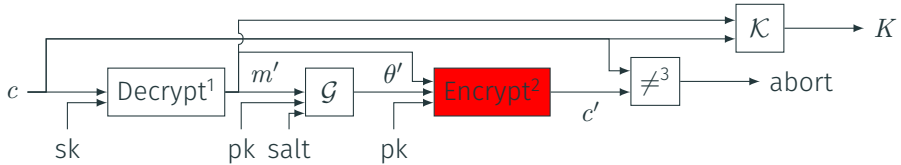


Figure 4: Decapsulation function

[1] WT+19; PT19.
[2] Guo+22.
[3] GJN20.

Disclosure

- February 2023: Disclosed the issue to the HQC design team.
- March 2023: Disclosed to PQClean
- 2024-02-21: We submitted our paper to ePrint Archive
- 2024-02-23: HQC team released a fixed optimized reference implementation, manually implementing the Barrett reductions.