Don’t Reject This: Key-Recovery Timing Attacks
Due to Rejection-Sampling in HQC and BIKE

Qian Guo³, Clemens Hlauschek¹,⁵, Thomas Johansson³,
Norman Lahr², Alexander Nilsson³,⁴, and Robin Leander Schröder¹
September 19, 2022

¹Technische Universität Wien, Austria
²Fraunhofer SIT, Darmstadt, Germany
³Lund University, Lund, Sweden
⁴Advenica AB, Malmö, Sweden
⁵RISE GmbH, Wien, Austria
Elevator Pitch

Rejection sampling with a seed derived from the message leaks the secret key.

Fundamentals
Attack
Countermeasures
Hamming-Quasi-Cyclic (HQC)
Code-based round 3 contender

Based on hard problems related to quasi-cyclic codes.
Key Encapsulation Mechanism (KEM) $E_{kem}$ with security parameter $\lambda$

Tuple of algorithms: $(\text{KeyGen}, \text{Encaps}, \text{Decaps})$

$(pk, sk) \leftarrow$ KeyGen($1^\lambda$)

$(k_0, c) \leftarrow$ Encaps($pk, 1^\lambda$)

$k_1 \leftarrow$ Decaps($sk, c$)

Correctness: $k_0 = k_1$ with overwhelming probability

Game Based Security:

IND-CPA: given encaps oracle, can’t distinguish real key from random key

IND-CCA: given additional decaps oracle
Hamming Quasi-Cyclic (HQC) PKE: Key Generation [Agu+16; Car+20]

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

KeyGen(param)

1. \( h, x, y \leftarrow \mathcal{R} \) with \( \omega(x) = \omega(y) = \omega \)
2. \( sk = (x, y) \)
3. \( pk = (h, s = x + h \cdot y) \)
4. \textbf{return} \((pk, sk)\)
HQC PKE: Encryption and Decryption

Encrypt(pk, m)
1 $e, r_1, r_2 \leftarrow \mathcal{R}$ with $\omega(e) = \omega_e$ and $\omega(r_1) = \omega(r_2) = \omega_r$
2 $u = r_1 + h \cdot r_2$
3 $v = mG + s \cdot r_2 + e$
4 \textbf{return } c = (u, v)

Decrypt(sk = (x, y), c = (u, v))
1 \textbf{return } C. \text{Decode}(v - u \cdot y)$
Decryption Failures

Decoding is successful when \( \mathbf{v} - \mathbf{u} \cdot \mathbf{y} \) has \( \leq \delta \) errors:

\[
\mathbf{v} - \mathbf{u} \cdot \mathbf{y} = mG + \mathbf{s} \cdot \mathbf{r}_2 + e - (r_1 + h \cdot \mathbf{r}_2) \cdot \mathbf{y} \\
= mG + (x + h \cdot \mathbf{y}) \cdot \mathbf{r}_2 + e - (r_1 + h \cdot \mathbf{r}_2) \cdot \mathbf{y} \\
= mG + x \cdot \mathbf{r}_2 + e - r_1 \cdot \mathbf{y} \\
\underbrace{\text{sparse}}_{} 
\]
HQC KEM: Key Generation and Encapsulation

KeyGen(param)

1 \textbf{return} \ PKE.\textit{KeyGen}(param)

Encaps(pk)

1 \( m \leftarrow \mathbb{F}_2^k \)
2 \( \theta = G(m) \)
3 \( c = \text{PKE.\textit{Encrypt}}(pk, m; \theta) \)
4 \( K = \mathcal{K}(m, c) \)
5 \( d = \mathcal{H}(m) \)
6 \textbf{return} \ (K, (c, d))
HQC KEM: Decaps

\[ c \rightarrow \text{Decrypt}^1 \rightarrow m' \rightarrow K \]

\[ d \]

\(^1\text{WT+19; PT19.} \]
\(^2\text{GJN20.} \]
HQC KEM: Decaps

\[ \begin{align*}
d & \quad \text{Decrypt}^1 \quad m' \quad G \quad \theta' \quad \text{Encrypt} \quad c' \\
& \quad \text{sk} \quad \text{pk} \\
& \quad \text{Abort}^2 \\
& \quad K \\
& \quad \mathcal{K}
\end{align*} \]

\[^1\text{WT+19; PT19.}\]
\[^2\text{GJN20.}\]
HQC KEM: Decaps

\[ \text{Decrypt}^1 \quad m' \quad G \quad \theta' \quad \text{Encrypt} \quad \neq^2 \quad \mathcal{V} \quad \text{abort} \]

\[ c \quad \text{sk} \quad \mathcal{K} \quad K \]

\[ d \]

---

\(^1\text{WT+19; PT19.} \)

\(^2\text{GJN20.} \)
HQC KEM: Decaps

\[ c \xrightarrow{\text{sk}} \text{Decrypt}^1 \xrightarrow{m'} G \xrightarrow{\theta'} \text{Encrypt} \xrightarrow{\neq^2} \mathcal{K} \xrightarrow{K} \]

\[ d \xrightarrow{} \mathcal{H} \xrightarrow{\neq} \] abort

\(^1\text{WT+19; PT19.}\)

\(^2\text{GJN20.}\)
HQC KEM: Decaps

\[ c \xrightarrow{\text{Decrypt}} \theta' \xrightarrow{\text{Encrypt}} c' \xrightarrow{\neq} \mathcal{V} \xrightarrow{\text{abort}} \]

\[ d \]

\[^1\text{WT+19; PT19.} \]

\[^2\text{GJN20.} \]
HQC KEM: Decaps

1WT+19; PT19.
2GJN20.
HQC KEM: Decaps

\[ \text{Decrypt}^1 \quad m' \quad G \quad \theta' \quad \text{Encrypt} \quad \neq^2 \quad \mathcal{V} \quad \text{abort} \]

\[ d \quad \mathcal{H} \quad \neq \quad c' \]

\[ c \quad \text{sk} \quad K \quad \mathcal{K} \]

\[ 1^{\text{WT+19; PT19.}} \]
\[ 2^{\text{GJN20.}} \]
Discovery of a Timing-Variation
Algorithm 6: Collecting timing measurements

1. \((pk, sk) \leftarrow \text{KeyGen}(1^n)\)
2. \(\text{for } i \in \{1, \ldots, \text{num\_ciphertexts}\} \text{ do}\)
3. \(\quad (c, k) \leftarrow \text{Encaps}(pk)\)
4. \(\quad \text{for } j \in \{1, \ldots, \text{num\_measurements}\} \text{ do}\)
5. \(\quad \quad \text{measure\_cycles(Decaps(sk, c))}\)
6. \(\quad \text{end}\)
7. \(\text{end}\)
Detecting timing differences

Figure 1: P-values of Welch’s t-test:
Statistically significant difference & No statistically significant difference.
Detected differences: 8260 cycles (≈ 4.13 μs @ 2 GHz).
Recursing into Decaps

(a) Key loading
205 cycles

(b) Decryption
462 cycles

(c) Re-encryption
7736 cycles

(d) Shared secret
92 cycles

Figure 2: P-values of Welch’s t-test
Recursing into Re-encryption

Figure 3: P-values of Welch’s t-test
Vector sampling

How to sample $e, r_1, r_2 \leftarrow \mathcal{R}$ in Encrypt:

Rejection sampling of a vector of length $n$ with Hamming weight $= w$. 
Algorithm 7: vect_set_random_fixed_weight

Input: weight \( w \), length \( n \leq 2^{24} \)

Result: vector \( v \) of length \( n \) with weight \( \|v\| = w \)

1. \( v = 0^n \)
2. \( \omega = 0 \)
3. repeat
   4. repeat
      5. \( i \leftarrow [0, 2^{24}) \)
      6. until \( i < \left\lfloor \frac{2^{24}}{n} \right\rfloor n \)
      7. \( i = i \mod n \)
      8. if \( v_i \neq 1 \) then
         9. \( v_i = 1 \)
         10. \( \omega = \omega + 1 \)
     end
   11. until \( \omega = w \)
4. return \( v \)
seedexpander(ctx, rand_bytes, random_bytes_size);
for (uint32_t i = 0 ; i < weight ; ++i) {
    do {
        if (j == random_bytes_size) {
            seedexpander(ctx, rand_bytes, random_bytes_size);
                \[only performed when randomess is exhausted\]
            j = 0;
        }
        random_data = ((uint32_t) rand_bytes[j++]) << 16;
        random_data |= ((uint32_t) rand_bytes[j++]) << 8;
        random_data |= rand_bytes[j++];
    } while (random_data >= UTILS_REJECTION_THRESHOLD);
    random_data = random_data % PARAM_N;
    // [...]
Figure 4: Data flow in HQC.
The message $m$ determines the timing of the ciphertext!

**Figure 4:** Data flow in HQC.
Figure 5: Timing distribution of decapsulation
Figure 6: Timing distribution of decapsulation
Attack
The message $m$ that a ciphertext decrypts to determines the timing of the message. The ciphertext does not have to be valid.
Prerequisites

The message $m$ that a ciphertext decrypts to determines the timing of the message.
The ciphertext does not have to be valid.

→ We can distinguish whether a modified ciphertext decrypts to a message $m$ or $m'$!
Recall: HQC encryption/decryption

Encrypt($pk, m$)

1. $e, r_1, r_2 \leftarrow \mathcal{R}$ with $\omega(e) = \omega_e$ and $\omega(r_1) = \omega(r_2) = \omega_r$
2. $u = r_1 + h \cdot r_2$
3. $v = mG + s \cdot r_2 + e$
4. return $c = (u, v)$

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

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

Set $r_1$ to 1 and $r_2$ and $e$ to 0 error is secret key!

Recover the error of the ciphertext to 🌟🌟win🌟🌟.

Additionally: we can add any extra error $e'$ we want, for a combined error of $e' - y$. 20
Using the distinguisher

Recall: ciphertexts do not have to be valid

Assume $\text{timing}(c_1) \neq \text{timing}(c_2)$

**Figure 7:** Random walk in ambient space $\mathbb{F}_2^n$ (symbolic image)
Flip bits until timing changes
Flip bits back to determine if they are an error
Repeat, take a majority vote
Evaluation

6096 attacks performed

Success rate: 87%

Among failed attacks: 86% terminated with less than 20 incorrect bits

866,143 idealized oracle calls (median)
BIKE Side-Channel and Attack
Algorithm 10:

BIKE.KeyGen

**Input:** ·

**Output:** $sk = (h_0, h_1, \sigma)$

$$pk = h \in \mathcal{R}$$

1. $(h_0, h_1) = \text{Sample}(\mathcal{H}_w)$
2. $h = h_1 h_0^{-1}$
3. $\sigma = \text{Sample}(\mathcal{M})$
4. $sk = (h_0, h_1, \sigma)$
5. $pk = h$
Algorithm 11: BIKE.Encaps

Input: \( pk = h \)
Output: \( K, c \)

1. \( m = \text{Sample}(\mathcal{M}) \)
2. \( (e_0, e_1) = H(m) \)
3. \( c = (e_0 + e_1, m \oplus L(e_0, e_1)) \)
4. \( K = K(m, c) \)

Algorithm 12: BIKE.Decaps

Input: \( sk = (h_0, h_1, \sigma) \)
\( c = (c_0, c_1) \)
Output: \( K \)

1. \( e' = \text{Decode}(c_0h_0, h_0, h_1) \)
2. \( m' = c_1 \oplus L(e') \)
3. if \( e' = H(m') \) then
   4. \( K = K(m', c) \)
4. else
   5. \( K = K(\sigma, c) \)
7. end
BIKE Side-Channel

![Graph showing the relationship between Num. PRNG Samplings and Num. Seedexpansions with clock cycles on the y-axis and Num. PRNG Samplings θ on the x-axis. The graph includes a legend for different seed expansions colors and a logarithmic scale for the x-axis.]
<table>
<thead>
<tr>
<th>Num. PRNG Samplings $\theta$</th>
<th>Num. Plaintexts</th>
<th>Num. Seedexpansions</th>
</tr>
</thead>
<tbody>
<tr>
<td>141</td>
<td>145</td>
<td>149</td>
</tr>
<tr>
<td>153</td>
<td>157</td>
<td>161</td>
</tr>
<tr>
<td>163</td>
<td>165</td>
<td>169</td>
</tr>
<tr>
<td>173</td>
<td>177</td>
<td>181</td>
</tr>
<tr>
<td>183</td>
<td>185</td>
<td>189</td>
</tr>
<tr>
<td>193</td>
<td>197</td>
<td>201</td>
</tr>
<tr>
<td>203</td>
<td>205</td>
<td>209</td>
</tr>
<tr>
<td>213</td>
<td>217</td>
<td>221</td>
</tr>
<tr>
<td>225</td>
<td>229</td>
<td>233</td>
</tr>
<tr>
<td>237</td>
<td>241</td>
<td></td>
</tr>
</tbody>
</table>
BIKE Attack

Reuse [GJS16] attack and [NJW18]

Observation: if the distance of an error occurs in the secret key, it lowers the decryption failure rate

Recover distance spectrum of the secret key with side-channel

Recover the secret key from the distance spectrum using a recursive-backtracking algorithm
Simplest version:

Ciphertext with rare timing behavior + added noise

Send ciphertext to timing oracle, check whether decoding failure occurred.

Derive whether a cyclic distance $d$ occurs in the secret key based on the decoding failure rate.
Countermeasures
Remove inner rejection sampling:

Sample a large number in steps, reduce modulo $n$
Determine a sufficient number of outer rejection sampling iterations.

“Sufficient”: will not require more iterations with overwhelming probability.

Perform fixed number of iterations.
Figure 8: Fixed version

But: heavy performance hit: +29% in cycle count.

Interesting alternative approaches: Constant-time and time-efficient Fisher-Yates

References


Don’t Reject This: Key-Recovery Timing Attacks Due to Rejection-Sampling in HQC and BIKE

Qian Guo\textsuperscript{3}, Clemens Hlauschek\textsuperscript{1,5}, Thomas Johansson\textsuperscript{3}, Norman Lahr\textsuperscript{2}, Alexander Nilsson\textsuperscript{3,4}, and Robin Leander Schröder\textsuperscript{1}

September 19, 2022

\textsuperscript{1}Technische Universität Wien, Austria
\textsuperscript{2}Fraunhofer SIT, Darmstadt, Germany
\textsuperscript{3}Lund University, Lund, Sweden
\textsuperscript{4}Advenica AB, Malmö, Sweden
\textsuperscript{5}RISE GmbH, Wien, Austria
Bonus Slides
Attack against RS/RM version

Figure 9: RS/RM Concatenated Code
Optimized Attack
Exploit the structure of the code generated by $G$.

The public code $C$ is either:

- a Bose-Chaudhuri-Hocquenghem (BCH) code tensored with a repetition code
- a Reed-Solomon (RS) code concatenated with a Reed-Muller (RM) code
We exploit the BCH/Repetition code version.

**Figure 10:** BCH/Repetition Tensor Code

Idea: corrupt $\delta$ BCH code blocks s.t. 1 more corruption will cause decoding failure. Then determine out the error in the repetition code block.
<table>
<thead>
<tr>
<th>Candidate</th>
<th>Action 1</th>
<th>Decode</th>
<th>Candidate</th>
<th>Action 2</th>
<th>Decode</th>
</tr>
</thead>
<tbody>
<tr>
<td>00010</td>
<td>⊕ 11011</td>
<td>= 11001 → 1 ✗</td>
<td>00010</td>
<td>⊕ 01000</td>
<td>= 01010 → 0 ✓</td>
</tr>
<tr>
<td>11000</td>
<td>⊕ 11011</td>
<td>= 00011 → 0 ✓</td>
<td>11000</td>
<td>⊕ 01000</td>
<td>= 10000 → 0 ✓</td>
</tr>
<tr>
<td>00100</td>
<td>⊕ 11011</td>
<td>= 11111 → 1 ✗</td>
<td>00100</td>
<td>⊕ 01000</td>
<td>= 01100 → 0 ✓</td>
</tr>
<tr>
<td>10001</td>
<td>⊕ 11011</td>
<td>= 01010 → 0 ✓</td>
<td>10001</td>
<td>⊕ 01000</td>
<td>= 11001 → 1 ✗</td>
</tr>
</tbody>
</table>

\[
\min(2\checkmark, 2\checkmark) = 2
\]

\[
\max(2, 1) = 2
\]

Select Action 1

\[
\min(3\checkmark, 1\checkmark) = 1
\]

\[
\max(2, 1) = 2
\]

Repetition Code Block \( \in \mathbb{F}_2^n \)

\( \delta \) Corrupted Blocks

Oracle \( \mathcal{O} \)

Remaining Candidates

\[
\begin{align*}
00010 & \times \\
11000 & \checkmark \\
00100 & \times \\
10001 & \checkmark \\
\end{align*}
\]
163 attacks performed
Success rate: 96.7%
Among failed attacks: less than 4 bits incorrect
19,942 idealized oracle calls (median)
Recover Distance Spectrum using the Side-Channel

Simplest version:

Construct a ciphertext with a message that has a rare timing-behavior and add an error to get close to the decoding limit\(^4\).

Send ciphertext to timing oracle, check whether decoding failure occurred.

Derive whether a cyclic distance \(d\) occurs in the secret key based on the decoding failure rate.

For each cyclic distance \(d\) in the error:

- If decoding success: increment \(\text{observed}_d\).
- If decoding failure: increment \(\text{failed}_d\).

For each distance \(d\), compute the empirical decoding failure rate, and estimate the multiplicity of the distance based on that.

\(^4\)Ciphertext does not have to be valid!
Distance Spectrum of a Vector

Vector $v$, length $r$.\(^5\)

Multi-set of cyclic distances between set bits in vector $v$.

$$v = 100001001$$

$$D(v) = \{\}$$

\(^5\)Graphics heavily inspired by https://youtu.be/Gm--Sm_wJ2w
Vector \( v \), length \( r \).\(^5\)

Multi-set of cyclic distances between set bits in vector \( v \).

\[
v = \begin{array}{cccccccc}
1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\
\end{array}
\]

\[D(v) = \{1\}\]

\(^5\)Graphics heavily inspired by https://youtu.be/Gm--Sm_wJ2w
Distance Spectrum of a Vector

Vector $v$, length $r$.\(^5\)

Multi-set of cyclic distances between set bits in vector $v$.

$$v = 100001001$$

$$D(v) = \{1, 4\}$$

\(^5\)Graphics heavily inspired by https://youtu.be/Gm--Sm_wJ2w
Distance Spectrum of a Vector

Vector $\mathbf{v}$, length $r$.\(^5\)

Multi-set of cyclic distances between set bits in vector $\mathbf{v}$.

\[ \mathbf{v} = 100001001 \]

\[ D(\mathbf{v}) = \{1, 3, 4\} \]

---

\(^5\)Graphics heavily inspired by https://youtu.be/Gm--Sm_wJ2w
Distances in the error affect the decoding failure rate

Satisfied parity checks during decoding\(^6\):

\[
\begin{align*}
\mathbf{h} &= \begin{bmatrix}
1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\
1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\
1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\
\end{bmatrix}, \\
\mathbf{s} &= \begin{bmatrix}
1 \\
0 \\
1 \\
0 \\
1 \\
1 \\
0 \\
0 \\
1 \\
0 \\
0 \\
0 \\
0 \\
0 \\
\end{bmatrix}, \\
\mathbf{e} &= \begin{bmatrix}
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{bmatrix}
\end{align*}
\]

\(^6\)Graphics heavily inspired by [https://youtu.be/Gm--Sm_wJ2w](https://youtu.be/Gm--Sm_wJ2w)
Recover Secret Key from Distance Spectrum

Greedy recursive-backtracking algorithm:

Start with empty vector $\mathbf{h} = 0^r$

Check if already done ($w$ bits already set, and $\mathbf{h}$ is the secret key)

For each bit position $i$

if all distances to $i$ exist in the distance spectrum

set bit $i$, and recurse