Key Recovery from Side-Channel Power Analysis Attacks on Non-SIMD HQC Decryption

Nathan Maillet [1,3] Cyrius Nugier[2] Vincent Migliore[3]
Jean-Christophe Deneuville [2]

[1] *EDF R&D, France*
[2] *Fédération ENAC ISAE-SUPAERO ONERA, France*
[3] *LAAS–CNRS / INSA-Toulouse, France*

Why would we want to remove the SIMD?

Usage

SIMD-less implementation represents a *portable*, *IoT-friendly* version, close to what would be used in production.

Why would we want to remove the SIMD?

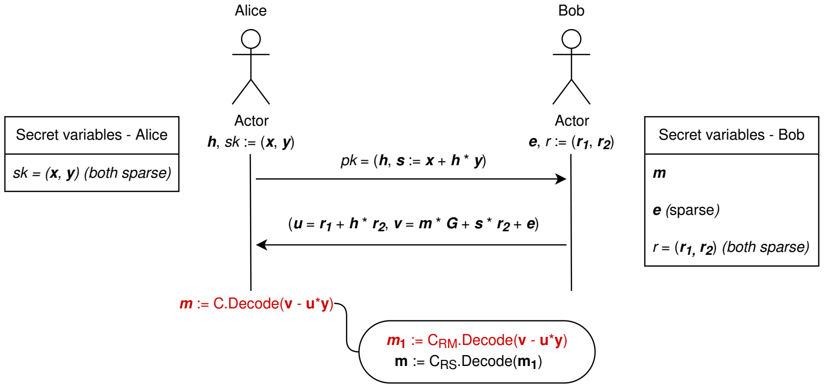
Usage

SIMD-less implementation represents a *portable*, *IoT-friendly* version, close to what would be used in production.

Version of HQC	Cycle count (mean)	Cycle count (std)
Optimized (SIMD `-O3`)	471	72
SIMD `-O2`	550	239
No SIMD	312	12
Reference	669	58

Table: Performances of the attacked function (`expand_and_sum`) for diverse implementations of HQC

Quick reminder of HQC


Alice
Actor
$\boldsymbol{h}$, $sk := (\boldsymbol{x}, \boldsymbol{y})$
Bob
Actor
$\boldsymbol{e}$, $r := (\boldsymbol{r_1}, \boldsymbol{r_2})$
Secret variables - Alice
$sk = (\boldsymbol{x}, \boldsymbol{y})$ (both sparse)
Secret variables - Bob
$\boldsymbol{m}$
$\boldsymbol{e}$ (sparse)
$r = (\boldsymbol{r_1}, \boldsymbol{r_2})$ (both sparse)
$pk = (\boldsymbol{h}, \boldsymbol{s} := \boldsymbol{x} + \boldsymbol{h} * \boldsymbol{y})$
$(\boldsymbol{u} = \boldsymbol{r_1} + \boldsymbol{h} * \boldsymbol{r_2}, \boldsymbol{v} = \boldsymbol{m} * \boldsymbol{G} + \boldsymbol{s} * \boldsymbol{r_2} + \boldsymbol{e})$
$\boldsymbol{m}$:= C.Decode($\mathbf{v}$ - $\mathbf{u}$*$\mathbf{y}$)
$\boldsymbol{m_1}$:= C_{RM}.Decode($\mathbf{v}$ - $\mathbf{u}$*$\mathbf{y}$)
$\mathbf{m}$:= C_{RS}.Decode($\mathbf{m_1}$)

Remarks

- HQC's code uses Reed-Muller (RM) and Reed-Solomon (RS) codes and works on vectors of Hamming Weight $\approx \sqrt{n}$, n being the length of the code.
- Any vector can be viewed as a polynomial of $\mathcal{R} = \mathbb{F}_2[X]/(X^n - 1)$.

Comparison of this work with known attacks on HQC

Attack	Oracle calls	HQC implem.	Type of SCA
[this work]	1	Optimized, no SIMD	⚡
[TCHES:GMGL24]	1	Reference	🧲
[USENIX:SchGasGuo24]	1142	Optimized	⏱
[TCHES:HSCGJ23; AC:GNNJ23]	9000	Reference	⏱
[TCHES:GHJLNS22; AC:GNNJ23]	10000	Optimized	⏱
[PQCRYPTO:SHRWS22]	52992	Reference	⚡
[TCHES:HSCGJ23]	53857	Reference	⏱
[TCHES:GHJLNS22]	866000	Optimized	⏱

Table: Oracle calls needed for each ISA-level attack on HQC-128

Two Side-Channel Attacks targeting the duplicated-RM decoding

An ISA-level attack needing only 1 oracle call

ISA	Optimization flag					
	-0g	-00	-01	-02	-03	-0z
RISC-V	V0/V1	V0/V1	V0/V1	V0/V1	V0/V1	-
AArch32	V0/V1	V0/V1	V0/V1	V0/V1	V0/V1	V0/V1
x86-64	V0/V1	V0/V1	V1	V1	V1	V1

Table: Applicability of the ISA-level attack for both of our variants

Two Side-Channel Attacks targeting the duplicated-RM decoding

An ISA-level attack needing only 1 oracle call

ISA	Optimization flag					
	−0g	−00	−01	−02	−03	−0z
RISC-V	V0/V1	V0/V1	V0/V1	V0/V1	V0/V1	-
AArch32	V0/V1	V0/V1	V0/V1	V0/V1	V0/V1	V0/V1
x86-64	V0/V1	V0/V1	V1	V1	V1	V1

Table: Applicability of the ISA-level attack for both of our variants

A replay attack on a Cortex-M4 microarchitecture

83 (resp. 56) traces are enough to ensure a key recovery with 99% (resp. 50%) for any HQC version.

Countermeasures to those attacks

Attack \ Countermeasure	Min. Dist. Decoding	Adding Noise	Codeword masking*
[this work]	✓	✓	✓
[TCHES:GMGL24]	×	×	✓
[USENIX:SchGasGuo24]	×	✓	✓
[PQCRYPTO:SHRWS22]	×	✓	×
[TCHES:HSCGJ23]	×	×	×
[TCHES:GHJLNS22]	×	×	×

Table: Effectiveness of our countermeasures for each known attack

*Under the assumption of a leakage-free codeword masking.

Code used in HQC

HQC's Encoding

$$\begin{tikzcd}
\text{Message} \arrow[rr, dashed, "\text{C.Encode()}"] \arrow[d] & & \text{Encoded Message} \\
\text{RS-encoded} \arrow[dr] & & \text{duplicated-RM-RS} \arrow[u] \\
& \text{RM-RS-encoded} \arrow[ur] &
\end{tikzcd}$$

Code used in HQC

HQC's Decoding

$$\begin{tikzcd}
\text{Message} & & \text{Encoded Message} \arrow[ll, dashed, "\text{C.Decode()}"'] \arrow[d] \\
\text{RS-encoded} \arrow[u] & & \text{duplicated-RM-RS} \arrow[dl, "\mathbf{v}-\mathbf{u}\cdot\mathbf{y}"] \\
& \text{RM-RS-encoded} \arrow[ul] &
\end{tikzcd}$$

From leakage to the secret key

Assuming $\mathbf{v} - \mathbf{u} \cdot \mathbf{y}$ is known by SCA, and $\mathbf{u}$ is invertible, we have:

$\mathbf{y} = \mathbf{u}^{-1} \cdot (\mathbf{v} - (\mathbf{v} - (\mathbf{u} \cdot \mathbf{y})))$, and thus, $\mathbf{x} = \mathbf{s} - \mathbf{h} \cdot \mathbf{y}$

From leakage to the secret key

Assuming $\mathbf{v} - \mathbf{u} \cdot \mathbf{y}$ is known by SCA, and $\mathbf{u}$ is invertible, we have:

$$\mathbf{y} = \mathbf{u}^{-1} \cdot (\mathbf{v} - (\mathbf{v} - (\mathbf{u} \cdot \mathbf{y}))), \text{ and thus, } \mathbf{x} = \mathbf{s} - \mathbf{h} \cdot \mathbf{y}$$

Theorem (Security reduction)

$\forall\, \mathbf{u}$, *a brute-force of* $n - \mathbf{rk}(\mathbf{rot}(\mathbf{u}))$ *bits is enough to recover* $\mathbf{y}$.

From leakage to the secret key

Assuming $\mathbf{v}-\mathbf{u}\cdot\mathbf{y}$ is known by SCA, and $\mathbf{u}$ is invertible, we have:

$$\mathbf{y}=\mathbf{u}^{-1}\cdot(\mathbf{v}-(\mathbf{v}-(\mathbf{u}\cdot\mathbf{y}))),\text{ and thus, }\mathbf{x}=\mathbf{s}-\mathbf{h}\cdot\mathbf{y}$$

Theorem (Security reduction)

$\forall\,\mathbf{u}$, *a brute-force of* $n-\mathbf{rk}(\mathbf{rot}(\mathbf{u}))$ *bits is enough to recover* $\mathbf{y}$.

Theorem (Specifity of $\mathbf{u}$)

Let $\mathbf{u}\in\mathcal{R}$ *and* $\mathbb{1}=1+X+\cdots+X^{n-1}$. *The following holds:*

$$\mathbf{rk}(\mathbf{rot}(u))=\begin{cases}0 & \textit{if } \mathbf{u}=0\\ 1 & \textit{if } \mathbf{u}=\mathbb{1}\\ n-1 & \textit{if } \mathbf{u}\neq 0 \textit{ and } HW(\mathbf{u}) \textit{ is even}\\ n & \textit{otherwise, i.e., if } \mathbf{u}\neq\mathbb{1} \textit{ and } HW(\mathbf{u}) \textit{ is odd}\end{cases}.$$

Building an oracle out of HQC's C implementation

```
typedef union {                typedef union {
  uint16_t u16[8];               int16_t i16[128];
} codeword;                    } expandedCodeword;

void expand_and_sum(expandedCodeword *dst, codeword src[]) {
  for (size_t part = 0; part < 8; part++) {
    for (size_t i = 0; i < 16; ++i) {
      dst->i16[(part << 4) + i] = src->u16[part] >> i & 1;
    }
  }
  // sum the rest of the copies
  ...
}
```

Figure: C code of the Reed-Muller `expand_and_sum` function's optimized version with no SIMD

From the C to the ASM: validation of the leakage

```
; Load from memory
lw $t_0$, 0($a_0$)
; Right shift of i
srl $t_1$, $t_0$, $r$
; Bit extraction
andi $t_1$, $t_1$, 1
; Store the extracted
   bit into memory
sh $t_1$, 0($a_1$)
```

Figure: Instruction flow of the bit extraction in `expand_and_sum`

From the C to the ASM: validation of the leakage

```
; Load from memory
lw t0, 0(a0)
; Right shift of i
srl t1, t0, r
; Bit extraction
andi t1, t1, 1
; Store the extracted
   bit into memory
sh t1, 0(a1)
```

Figure: Instruction flow of the bit extraction in `expand_and_sum`

Extraction at i = 14

inst	t_1				
srl-	0	...	0	0	b_{14}
srl+	0	...	0	b_{15}	b_{14}

From the C to the ASM: validation of the leakage

```
; Load from memory
lw t0, 0(a0)
; Right shift of i
srl t1, t0, r
; Bit extraction
andi t1, t1, 1
; Store the extracted
    bit into memory
sh t1, 0(a1)
```

Figure: Instruction flow of the bit extraction in `expand_and_sum`

Extraction at i = 14

inst	t_1				
srl-	0	...	0	0	b_{14}
srl+	0	...	0	b_{15}	b_{14}

Extraction at i < 14

inst	t_1				
srl-	0	...	0	0	b_{i-1}
srl+	0	...	0	0	b_i

Easiness of real-world microarchitecture

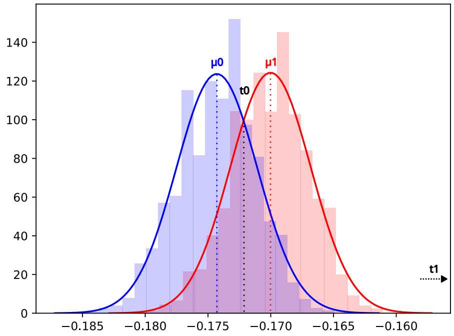


Figure: Power distributions for 0 and 1 on the point of interest for bit_0

Attack on Cortex-M4

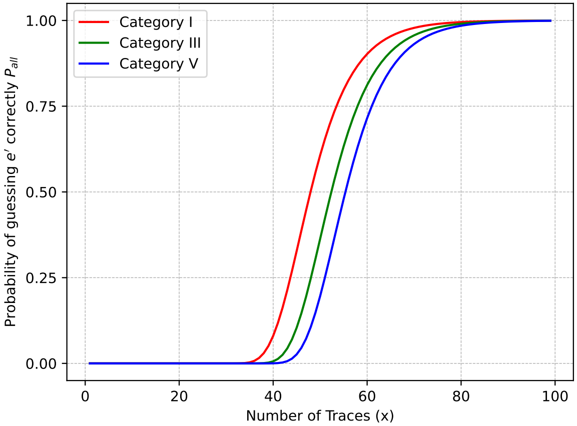


Figure: Probability of success of $(\mathbf{v} - \mathbf{u} \cdot \mathbf{y})$'s retrieval

Idea behind each countermeasure

Min. dist. decoder

Idea: Compare all (256) codewords to find the nearest to the input using the Hamming Distance.

As a result, an attacker can't peek on the result of $\mathbf{v} - \mathbf{u} \cdot \mathbf{y}$ as the `expand_and_sum` function does not exists anymore.

Idea behind each countermeasure

Min. dist. decoder

Idea: Compare all (256) codewords to find the nearest to the input using the Hamming Distance.

Adding noise

Idea: Add a random error $\mathbf{e}''$ of weight $\mathbf{w}_{\mathbf{e}''}$ such that $\binom{n}{\mathbf{w}_{\mathbf{e}''}} \gg 2^{128}$ to $\mathbf{v}$ before decoding.

As a result, peeking on the `expand_and_sum` function gives $(\mathbf{v} - \mathbf{u} \cdot \mathbf{y}) + \mathbf{e}'' = (\mathbf{mG} + \mathbf{e}') + \mathbf{e}''$.

Cat.	$\overline{HW(\mathbf{e}')}$	$w_{\mathbf{e}''}$	$\overline{HW(\mathbf{e}' + \mathbf{e}'')}$	DFR	DFR'
I	6003.93	11	6007.45	-132.86	-132.24

Idea behind each countermeasure

Min. dist. decoder

Idea: Compare all (256) codewords to find the nearest to the input using the Hamming Distance.

Adding noise

Idea: Add a random error $\mathbf{e}''$ of weight $\mathbf{w}_{\mathbf{e}''}$ such that $\binom{n}{\mathbf{w}_{\mathbf{e}''}} \gg 2^{128}$ to $\mathbf{v}$ before decoding.

Codeword masking

Idea: Encode a new (random) message $\mathbf{m}'$ and compute $\mathbf{v} + \mathbf{m}'\mathbf{G}$ before decoding.
As a result, reproducing the attacks “uselessly leaks” $\mathbf{v} - \mathbf{u} \cdot \mathbf{y} + \mathbf{m}'\mathbf{G}$.

Sum-up of the countermeasures

	Proposed countermeasures to prevent our attacks							
Min. dist. decoder	×	✓	×	×	✓	×	✓	✓
Adding noise	×	×	✓	×	✓	✓	×	✓
Codeword masking	×	×	×	✓	×	✓	✓	✓
Cycles ($\times 10^3$)	62875	66232	62963	63589	66320	63660	66929	67017
Overhead	0%	+5.34%	+0.14%	+1.14%	+5.48%	+1.25%	+6.45%	+6.59%

Table: Comparison to the reference implementation of the overhead of each countermeasure

Thank you for your attention

Acknowledgments

This work was partially supported by the Institut Cybersecurité Occitanie (ICO) and the joint-lab SEIDO with the active participation of Arthur Villard, EDF R&D, France.

Bibliography I

[TCHES:GMGL24] Goy, G., Maillard, J., Gaborit, P., Loiseau, A.
Single trace HQC shared key recovery with SASCA.
IACR Transactions on Cryptographic Hardware and Embedded Systems 2024(2), 64–87 (2024).

[USENIX:SchGasGuo24] Schröder, R.L., Gast, S., Guo, Q.
Divide and surrender: Exploiting variable division instruction timing in HQC key recovery attacks.
USENIX Security 2024: 33rd USENIX Security Symposium

[TCHES:HSCGJ23] Huang, S., Sim, R.Q., Chuengsatiansup, C., Guo Q., Johansson, T.
Cache-timing attack against HQC.
IACR Transactions on Cryptographic Hardware and Embedded Systems 2023(3), 136–163 (2023)

Bibliography II

[TCHES:GHJLNS22] Guo, Q., Hlauschek, C., Johansson, T., Lahr, N., Nilsson, A., Schröder, R.L.
Don't reject this: Key-recovery timing attacks due to rejection-sampling in HQC and BIKE.
IACR Transactions on Cryptographic Hardware and Embedded Systems 2022(3), 223–263 (2022)

[PQCRYPTO:SHRWS22] Schamberger, T., Holzbaur, L., Renner, J., Wachter-Zeh, A., Sigl, G.
A power side-channel attack on the reed-muller reed-solomon version of the HQC cryptosystem.
Post-Quantum Cryptography - 13th International Workshop, PQCrypto 2022. pp. 327–352

Bibliography III

[AC:GNNJ23] Guo, Q., Nabokov, D., Nilsson, A., Johansson, T.
SCA-LDPC: A code-based framework for key-recovery side-channel attacks on post-quantum encryption schemes.
Advances in Cryptology – ASIACRYPT 2023, Part IV. Lecture Notes in Computer Science, vol. 14441, pp. 203–236

[TIT:ABDGZ18] Aguilar-Melchor, C., Blazy, O., Deneuville, JJ.C. and Gaborit, P. and Zémor, G.
Efficient encryption from random quasi-cyclic codes.
IEEE Transactions on Information Theory 64(5), 3927–3943 (2018)

Bibliography IV

[DCC:AADGLZ24] Aguilar-Melchor, C., Aragon, N., Deneuville, J.C., Gaborit, P., Lacan, J., Zémor, G.
Efficient error-correcting codes for the HQC post-quantum cryptosystem.
Designs, Codes and Cryptography 92(12), 4511–4530 (2024)

[NIST:HQCround4] NIST
Post-Quantum Cryptography PQC | CSRC
https://csrc.nist.gov/projects/post-quantum-cryptography/selected-algorithms