High-Order and Cortex-M4 First-Order Implementations of Masked FrodoKEM

François Gérard[1] and Morgane Guerreau[2]*

[1]University of Luxembourg, [2]PQShield

Introduction

The recent standardization of post-quantum schemes following the NIST standardization process has shown that the field is maturing and calls for further evolutions on the practical side. In particular, side-channel secure implementations have to be developed on various platforms. Here, we will focus on the masking countermeasure for FrodoKEM, a KEM recommended by European agencies ANSSI and BSI that is a plain LWE sibling of the standard ML-KEM. While the masking techniques required to protect different parts of the scheme have already appeared in the literature, we make the first step toward securely deploying the scheme by proposing a high-order generic C implementation. Furthermore, we specialize the code at order 1 for Cortex-M4, by rewriting in ARM assembly the basic masking gadgets used by the generic implementation, in order to thwart (micro-)architectural leakage. Our work is validated by performing TVLA on a ChipWhisperer-Lite.

FrodoKEM Decaps

Input: $c = c_1 \| c_2 \| salt,\ sk = s \| seed_A \| b \| \mathbf{S}^T$
Output: ss

$\mathbf{B}' \leftarrow \mathsf{Unpack}(c_1, \bar{n}, n)$
$\mathbf{C} \leftarrow \mathsf{Unpack}(c_2, \bar{n}, \bar{n})$
$\mathbf{M} \leftarrow \mathbf{C} - \mathbf{B}'\mathbf{S}$
$u' \leftarrow \mathsf{Decode}(\mathbf{M})$
$seed_{SE'} \| k' \leftarrow \mathsf{SHAKE}(u' \| salt)$
$\mathbf{S}', \mathbf{E}', \mathbf{E}'' \leftarrow \mathsf{SampleMatrix}(seed_{SE'})$
$\mathbf{A} \leftarrow \mathsf{Gen}(seed_A)$
$\mathbf{B}'' \leftarrow \mathbf{S}'\mathbf{A} + \mathbf{E}'$
$\mathbf{B} \leftarrow \mathsf{Unpack}(b, n, \bar{n})$
$\mathbf{V} \leftarrow \mathbf{S}'\mathbf{B} + \mathbf{E}''$
$\mathbf{C}' \leftarrow \mathbf{V} + \mathsf{Encode}(u')$
$\bar{k} \leftarrow k'$ **if** $\mathbf{B}' \| \mathbf{C} = \mathbf{B}'' \| \mathbf{C}'$ **else** $\bar{k} \leftarrow s$
$ss \leftarrow \mathsf{SHAKE}(c_1 \| c_2 \| salt \| \bar{k})$
return ss

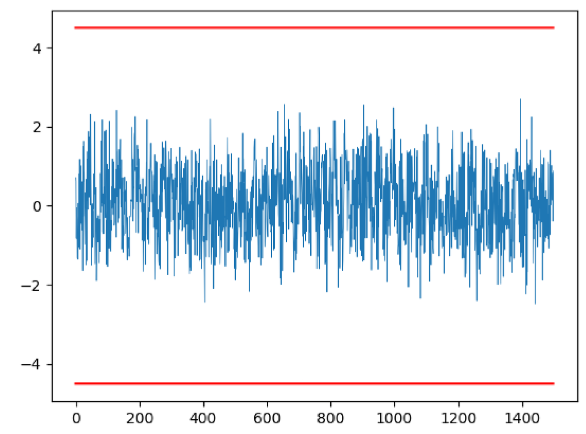
What to mask and how?

- Matrix operations: Use the fact that $\mathbf{AS}$ can be computed as $\mathbf{AS}_0, \mathbf{AS}_1, \ldots, \mathbf{AS}_n$.
- Hashing: Required for the re-encryption through the FO transform. Out of scope of this work 😇
- Encoding and decoding: Encode maps $\bar{n}^2$ B-bit substrings to values in $\mathbb{Z}_q$ by multiplying them by $q/2^B$. Decode is the inverse. Since q is a power of two, these are shifts, and we use A2B/B2A conversions to compute them.
- Gaussian sampling: Go through a cumulative distribution table. Most expensive operation in masked form.
- Comparison: To verify if a is equal to b, we compute $a - b$ and perform a masked zero test.

Micro-architectural Leakage

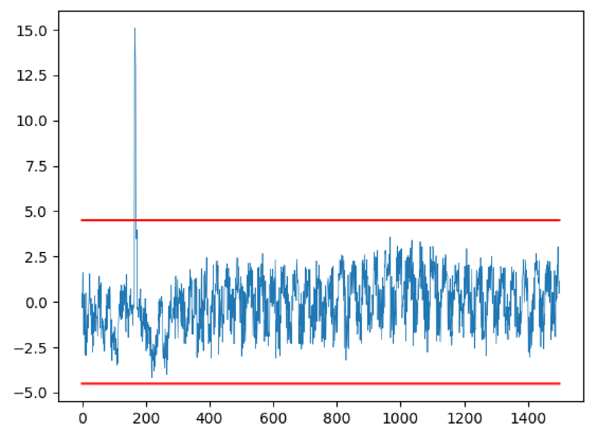
While implementing a theoretically validated masking scheme is a good first step toward a secure implementation, it is known that micro-architectural effects can introduce additional sources of leakage by, for example, manipulating shares on a common resource (register, bus). It is thus required to work at low-level to aim at limiting those effects.

```
ldrh rx0, [px], #2
ldrh r, [pool]
ldrh rx1, [px], #2
```



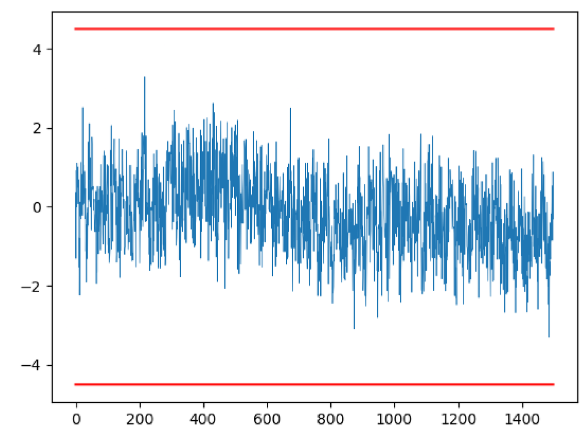
```
ldrh rx0, [px], #2
ldrh r, [pool]
ldrh rx1, [px], #2

strh t1, [pool]
```



```
ldrh rx0, [px], #2
ldrh r, [pool]
ldrh rx1, [px], #2

eor t2, t2, t2
strh t1, [pool]
```



Experimental validation

TVLA for naive (resp. hardened) ASM versions using 5,000 (resp. 100,000) traces.

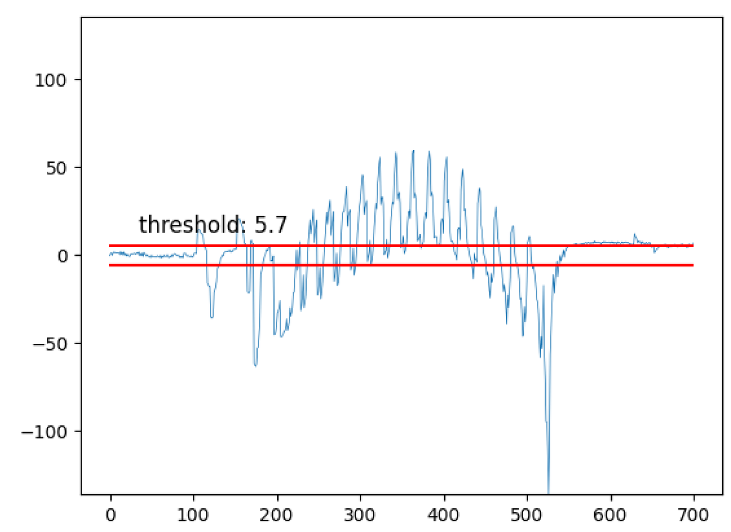


Fig. 1: t-tests for SecADD (naive)

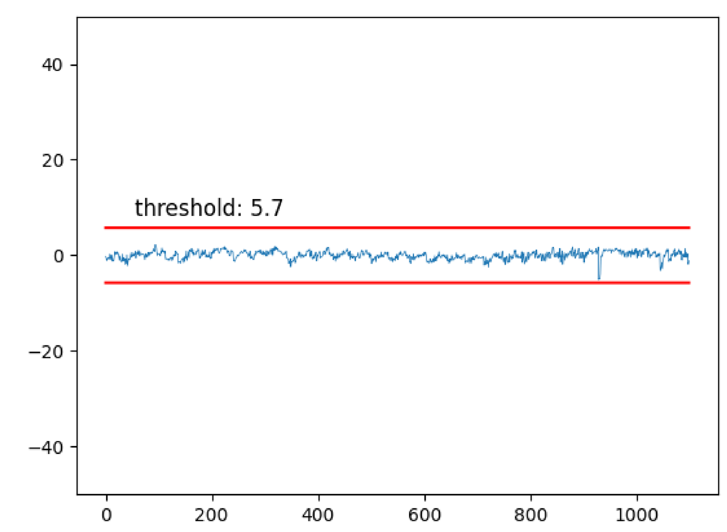


Fig. 2: t-tests for SecADD (hardened)

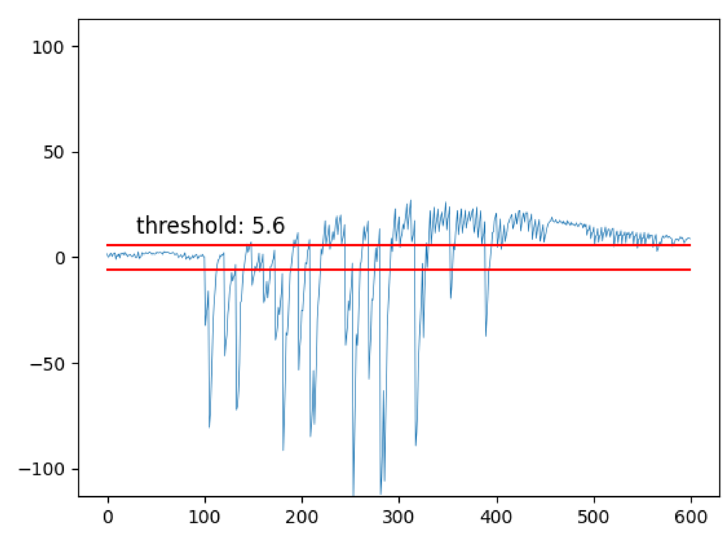


Fig. 3: t-tests for SecZeroTest (naive)

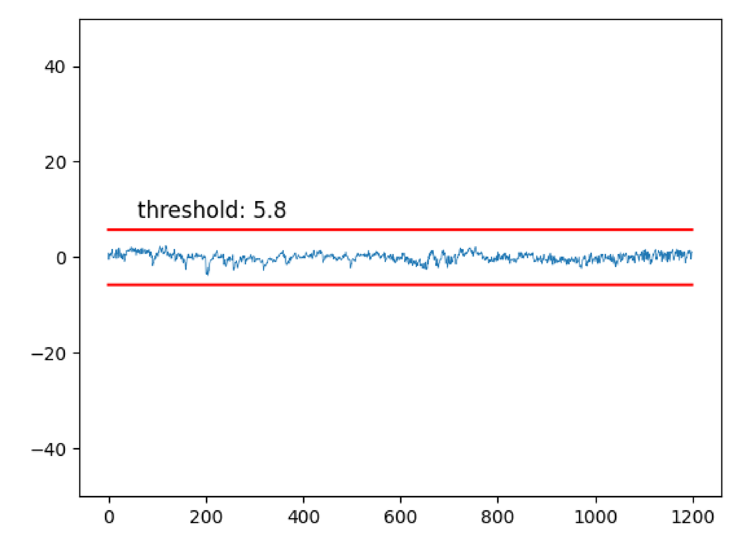


Fig. 4: t-tests for SecZeroTest (hardened)

Overhead on Cortex-M4

	C	ASM	ASM$_h$
SecAND	49	51	68 (+39%)
SecADD	206	149	248 (+20%)
BooleanToArithmetic	33	47	54 (+64%)
ArithmeticToBoolean	154	125	222 (+44%)
SecZeroTest	144	126	223 (+55%)

Tab. 1: Number of cycles on Cortex-M4

Manually hardening the gadgets induces a performance overhead ranging from +20% to +64%. This performance loss remains better than what can be obtained with automated tools.

Performances on x64

Order	1	2	3	4	5	6	7
Key encode	2	152	219	392	596	788	994
Key decode	4	76	113	234	368	483	602
Compare	408	6312	9370	19,358	31,125	40,369	49,756
Sampler	9647	94,967	144,200	223,058	318,535	422,430	531,283

Tab. 2: Benchmarks on x64 of the large gadgets used in FrodoKEM (in kilocycles)

Order	1	2	3	4	5	6	7
Frodo-640	57,395	293,703	498,200	809,345	1,138,708	1,523,870	2,010,644
Frodo-976	93,498	446,945	781,157	1,258,737	1,844,603	2,490,802	3,143,712
Frodo-1344	117,860	522,497	903,360	1,550,615	2,215,093	2,962,697	3,874,540

Tab. 3: Benchmarks on x64 of all versions of FrodoKEM (in kilocycles).

ePrint

ia.cr/2025/1065

Randomness usage

Order	1	2	3	4	5	6	7
Frodo-640	855	9388	18,632	31,628	47,673	66,720	88,816
Frodo-976	1113	12,524	24,861	42,336	63,884	89,434	119,058
Frodo-1344	1013	12,392	24,611	42,342	64,121	89,850	119,628

Tab. 4: Number of random 16-bit integers required during decaps (×1000)

Even if they have different parameters, Frodo-1344 and Frodo-976 actually need roughly the same amount of randomness because the CDT of Frodo-1344 is smaller.

GitHub

fragerar/masked_Frodo

* Work done while the author was at CryptoNext Security