SCA-LDPC: A Code-Based Framework for Key-Recovery Side-Channel Attacks on Post-Quantum Encryption Schemes

Qian Guo[1] Denis Nabokov[1] Alexander Nilsson[1,2] Thomas Johansson[1]

December 6, 2023

[1]Dept. of Electrical and Information Technology, Lund University, Lund, Sweden
{qian.guo,denis.nabokov,alexander.nilsson,thomas.johansson}@eit.lth.se

[2]Advenica AB, Malmö, Sweden

Table of contents

1. Introduction
2. Coefficient-wise Key-Recovery
3. Another approach to oracles
4. SCA-LDPC framework
5. Applications

Introduction

Framework for Side-Channel Attacks

SCAs are designed to break crypto in the presence of additional information

Fewer traces ⇒ more powerful attack

We propose a general framework to reduce the number of traces required for key-recovery on post-quantum KEMs

Apply a framework for Kyber and HQC

- Kyber — lattice-based primary KEM algorithm for standardization
- HQC — perspective code-based candidate in round 4

Framework for Side-Channel Attacks

SCAs are designed to break crypto in the presence of additional information

Fewer traces $\Rightarrow$ more powerful attack

We propose a general framework to reduce the number of traces required for key-recovery on post-quantum KEMs

Apply a framework for Kyber and HQC

- Kyber — lattice-based primary KEM algorithm for standardization
- HQC — perspective code-based candidate in round 4

Framework for Side-Channel Attacks

SCAs are designed to break crypto in the presence of additional information

Fewer traces $\Rightarrow$ more powerful attack

We propose a general framework to reduce the number of traces required for key-recovery on post-quantum KEMs

Apply a framework for Kyber and HQC

- Kyber — lattice-based primary KEM algorithm for standardization
- HQC — perspective code-based candidate in round 4

Framework for Side-Channel Attacks

SCAs are designed to break crypto in the presence of additional information

Fewer traces ⇒ more powerful attack

We propose a general framework to reduce the number of traces required for key-recovery on post-quantum KEMs

Apply a framework for Kyber and HQC

- Kyber — lattice-based primary KEM algorithm for standardization
- HQC — perspective code-based candidate in round 4

Oracles

- In (code-based and lattice-based) KEM decapsulation involves obtaining a message
- m' is connected to sk for the special ciphertext

Algorithm 1 KEM based on FO transform

Input: $c, \mathsf{sk} = (s_1, \ldots, s_k), \mathsf{pk}, z$

Function DECAPS($c, \mathsf{sk}, \mathsf{pk}, z$)
    $m' \leftarrow \mathsf{PKE.Dec}(\mathsf{sk}, c)$
    $r' \leftarrow \mathsf{G}(m'[, \mathsf{pk}])$
    $c' \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}, m', r')$
    if $c = c'$ **then**
        return $\mathsf{H}(m', c)$
    else
        return $\mathsf{H}_{\mathsf{prf}}(z, c)$

Side-channel-assisted CCA using Side-Channel Oracle[1] that leaks information about m'

[1]Such as plaintext-checking, decryption-failure, full-domain, etc.

Oracles

- In (code-based and lattice-based) KEM decapsulation involves obtaining a message
- *m′* is connected to sk for the special ciphertext

Algorithm 1 KEM based on FO transform

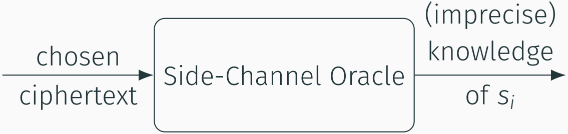
Input: $c, \mathsf{sk} = (s_1, \ldots, s_k), \mathsf{pk}, z$

1: **Function** DECAPS$(c, \mathsf{sk}, \mathsf{pk}, z)$
2: $\quad m' \leftarrow \mathsf{PKE.Dec}(\mathsf{sk}, c)$
3: $\quad r' \leftarrow \mathsf{G}(m'[, \mathsf{pk}])$
4: $\quad c' \leftarrow \mathsf{PKE.Enc}(\mathsf{pk}, m', r')$
5: $\quad$ **if** $c = c'$ **then**
6: $\quad\quad$ **return** $\mathsf{H}(m', c)$
7: $\quad$ **else**
8: $\quad\quad$ **return** $\mathsf{H}_{\mathsf{prf}}(z, c)$

Side-channel-assisted CCA using Side-Channel Oracle[1] that leaks information about *m′*

[1]Such as plaintext-checking, decryption-failure, full-domain, etc.

Oracles (cont.)

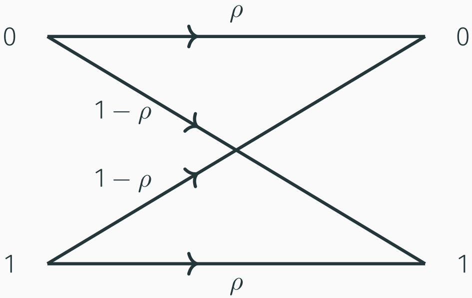


An oracle hides timing, cache-timing, power, electromagnetic, etc. leakages

Oracles (cont.)

The oracle is inherently inaccurate

Typically, oracle with accuracy ρ behaves similarly to Binary Symmetric Channel[2]



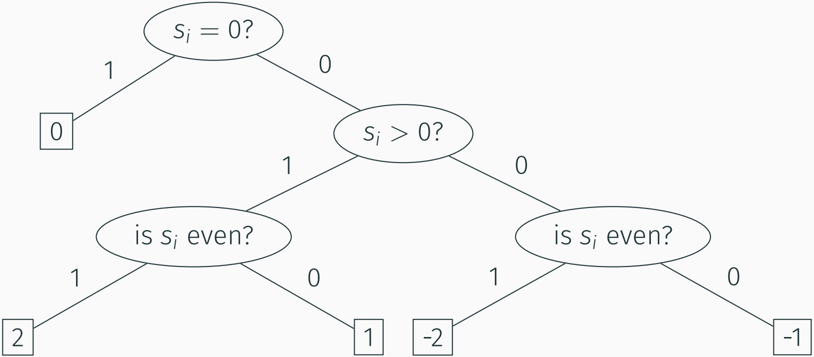
[2]Advanced oracle may return a bit with probability/confidence to be correct

Coefficient-wise Key-Recovery

Example of coefficient retrieval

Binary case: single oracle call reveals the coefficient

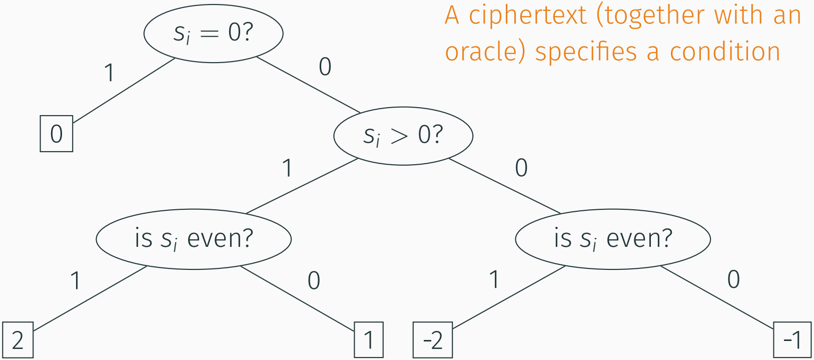
In Kyber-768, a secret coefficent $s_i \in \{-2, -1, 0, 1, 2\}$



Example of coefficient retrieval

Binary case: single oracle call reveals the coefficient

In Kyber-768, a secret coefficent $s_i \in \{-2, -1, 0, 1, 2\}$

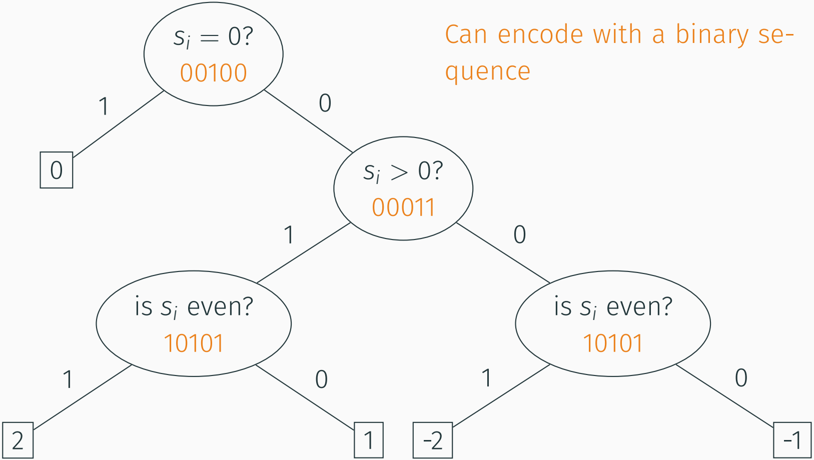


A ciphertext (together with an oracle) specifies a condition

Example of coefficient retrieval

Binary case: single oracle call reveals the coefficient

In Kyber-768, a secret coefficent $s_i \in \{-2, -1, 0, 1, 2\}$



Can encode with a binary sequence

Imperfect oracle

Problem

The probability to recover secret key with a real-world oracle is very low

Common solution

Repeat the same ciphertext several times

Majority voting gives more accurate oracle

This approach requires a lot of traces to achieve adequate probability of key recovery

Another approach to oracles

Working on (empirical) distributions

Each oracle call updates the (known) distribution of s_i

An oracle (BSC) with accuracy $\rho = 0.95$

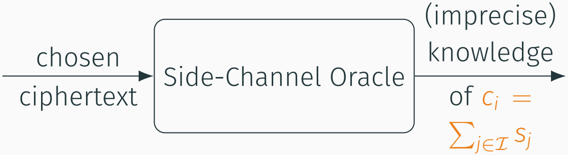
$$\begin{array}{ccc} \text{Prior} & & \text{Posterior} \\ \mathbf{Pr}[s_i = 0] = 0.9 & \xrightarrow[\text{returns } 1]{\text{oracle}} & \mathbf{Pr}[s_i = 0] = 0.32 \\ \mathbf{Pr}[s_i = 1] = 0.1 & & \mathbf{Pr}[s_i = 1] = 0.68 \end{array}$$

Framework works with empirical distributions

Our approach: do NOT try to focus all probability mass on a single value, some uncertainty for s_i is okay

Check variables

Can construct a ciphertext connecting to $s_i \overset{\text{with some}}{\underset{\text{modifications}}{\Longrightarrow}}$
Can construct a ciphertext connecting to $c_i = \sum_{j \in \mathcal{I}} s_j$



Similarly, can use a few traces to update distribution of c_i

Benefits of check variables

Source coding

HQC case (binary secret)

HQC has a sparse secret key, each coefficient s_j can be approximated to be from the Bernoulli distribution. With the perfect oracle, the obtained information for a bit s_j is bounded by 0.0352 bit for hqc-128. Obtaining a bit for check variable as a XOR of 50 coefficients gives 0.6255 bit of information.

Error correction — use checks to correct the distributions of secret coefficients

SCA-LDPC framework

LDPC code

Low-density parity-check (LDPC) code — linear code with a sparse parity-check matrix

Why LDPC?

- close to optimal error correction performance
- efficient decoding

$$\mathbf{H} = \left[\mathbf{H}_{r \times k} \middle| - \mathbf{I}_{r \times r} \right]^{3}$$

- k secret positions to recover
- r parity checks (variables)

[3] $\mathbf{H}_{r \times k}$ is a sub-matrix of a matrix consisting of blocks of circulant (or negacyclic) matrices

LDPC code

Low-density parity-check (LDPC) code — linear code with a sparse parity-check matrix

Why LDPC?

- close to optimal error correction performance
- efficient decoding

$$\mathsf{H} = \left[\mathsf{H}_{r \times k} | - \mathsf{I}_{r \times r}\right]^{3}$$

- k secret positions to recover
- r parity checks (variables)

[3] $\mathsf{H}_{r \times k}$ is a sub-matrix of a matrix consisting of blocks of circulant (or negacyclic) matrices

LDPC code (cont.)

Each row of $\mathbf{H}_{r\times k}$ defines a check variable $c_{k+1}, \ldots, c_{k+r}$

$$\left[\mathbf{H}_{r\times k} | -\mathbf{I}_{r\times r}\right] \cdot \left[s_1 \ldots s_k | c_{k+1} \ldots c_{k+r}\right]^{\mathsf{T}} = \mathbf{0}$$

We compute empirical distributions for $s_1, \ldots, s_k$ and $c_{k+1}, \ldots, c_{k+r}$. Send them to LDPC decoder

Decoder = iterative decoding via Belief Propagation using soft information

Output: updated distributions for $s_1, \ldots, s_k$ (i.e. error-corrected)

Attack steps

1. Choose r, create a “good” matrix $\mathbf{H}$
2. (Adaptively) call an oracle a few (could be 0) times to get empirical distribution for s_j
3. Similarly for c_i[4]
4. Call LDPC decoder with empirical distributions for $s_1, \ldots, s_k, c_{k+1}, \ldots, c_{k+r}$, obtain updated distributions for $s_1, \ldots, s_k$
5. Output hard values for $s_1, \ldots, s_k$

[4] Number of oracle calls for s_j and c_i is usually different

Applications

Kyber results

We attack masked implementation of Kyber-768 for ARM Cortex-M4[5]

We use ChipWhisperer toolkit to run power analysis

The NN model is trained on a profiling device, then applied to the attacked device

A non-adaptive power attack with a full-domain oracle (returns the whole decrypted message)

- 1 power trace = information about 256 secret coefficients/check variables

[5] *https://github.com/masked-kyber-m4/mkm4*

Kyber results

We attack masked implementation of Kyber-768 for ARM Cortex-M4[5]

We use ChipWhisperer toolkit to run power analysis

The NN model is trained on a profiling device, then applied to the attacked device

A non-adaptive power attack with a full-domain oracle (returns the whole decrypted message)

- 1 power trace = information about 256 secret coefficients/check variables

[5] *https://github.com/masked-kyber-m4/mkm4*

Kyber results

We attack masked implementation of Kyber-768 for ARM Cortex-M4[5]

We use ChipWhisperer toolkit to run power analysis

The NN model is trained on a profiling device, then applied to the attacked device

A non-adaptive power attack with a full-domain oracle (returns the whole decrypted message)

- 1 power trace = information about 256 secret coefficients/check variables

[5] *https://github.com/masked-kyber-m4/mkm4*

Kyber results (cont.)

Average accuracy across message bits is about 0.95

	Number of traces	Average number of errors[6]
Majority Voting	99	0.34/768
Our Method	12	0.82/768

Theoretical lower bound is 9 traces

[6] light post-processing is allowed

HQC results

An adaptive timing attack with a plaintext-checking oracle (tells if the decrypted message is equal to some fixed message)

No traces for secret coefficients, only for check variables as a XOR of 50 coefficients

	Number of "Idealized oracle"[7] calls
[16][8]	866000
Our Method	≈ 10000

[7] Noise-free environment, but not 100% correct

[8] 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.

Conclusion

- Proposed a framework to significantly reduce the number of traces for successful key recovery
  - Soft information/empirical distributions
  - Check variables
  - LDPC decoder (Belief Propagation)
- Showed real-world benefits for Kyber and HQC

Future work:

- Automate the selection of parameters (i.e. number of check variables, number of oracle calls, etc.)
- More advanced code-construction method with improved decoding performance
- Heavy post-processing (lattice-reduction or information-set decoding)

Questions?

Kyber simulation

Table 1: Comparison with the majority voting for full-key recovery. t is the number of votes cast, values in the brackets are m_0, m_1 and m_2, resp.

$\rho = 0.995$	Number of traces	Average number of errors
Majority Voting ($t = 3$)	27 (ref)	0.21/768
Our Method $(2, 1, 4)$	10 (-63%)	0.37/768
$\rho = 0.95$	Number of traces	Average number of errors
Majority Voting ($t = 7$)	63 (ref)	0.47/768
Our Method $(3, 4, 2)$	17 (-73%)	0.16/768
$\rho = 0.9$	Number of traces	Average number of errors
Majority Voting ($t = 11$)	99 (ref)	0.67/768
Our Method $(4, 3, 4)$	24 (-75.8%)	0.46/768

Maximizing information (lattice-based schemes)

For each s_i we call an oracle a few times with different ciphertexts. How to choose them?

Can choose ciphertexts maximizing the information (difference between entropies) gain

The Shannon's binary entropy function

$H(X) = -\sum_{x \in \mathcal{X}} \Pr[X = x] \log_2 \Pr[X = x]$