CENTER FOR
CYBER DEFENCE AND
INFORMATION SECURITY

How AI Can Extract Secrets from Electronic Chips

Elena Dubrova, KTH

CDIS Spring Conference 2023
KTH Royal Institute of Technology
Thursday, May 25, 2023

Outline

- Background
  - Side-channel analysis
  - Masking countermeasure
  - CRYSTALS-Kyber KEM post-quantum cryptogrphic algorithm
- Side-channel attack on a higher-order masked CRYSTALS-Kyber implementation
  - Breaking a Fifth-Order Masked Implementation of CRYSTALS-Kyber by Copy-Paste, E. Dubrova, K. Ngo, J. Gärtner R. Wang, *Real World Crypto Symposium, March 2023*, *https://eprint.iacr.org/2022/1713*
- Summary & future work

Side-channel attacks

- Algorithms are implemented in MCUs, CPUs, FPGAs, ASICs
- Different operations may consume different amount of power/time
- The same operation executed on different data may consume different amount of power/time
- It may be possible to recognize which operations and data are processed from power/time

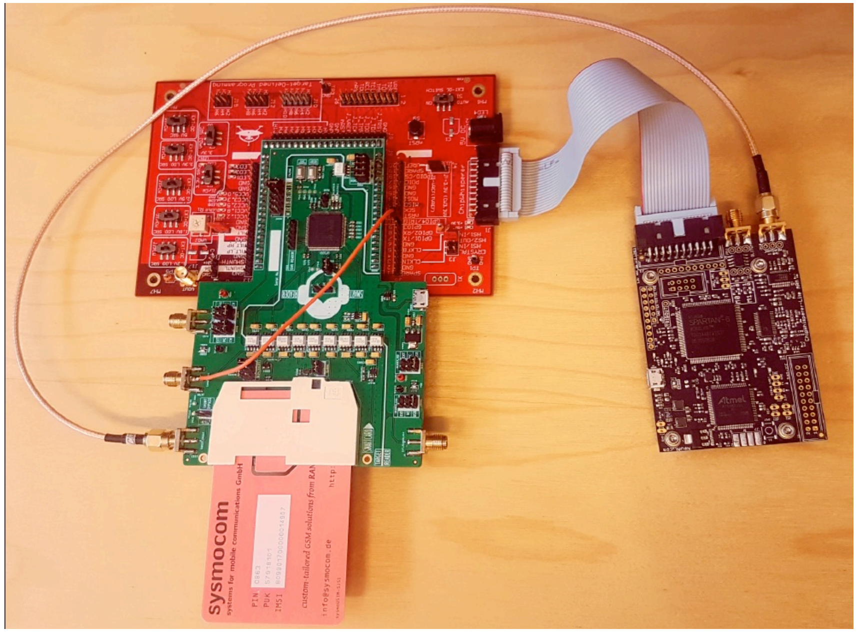
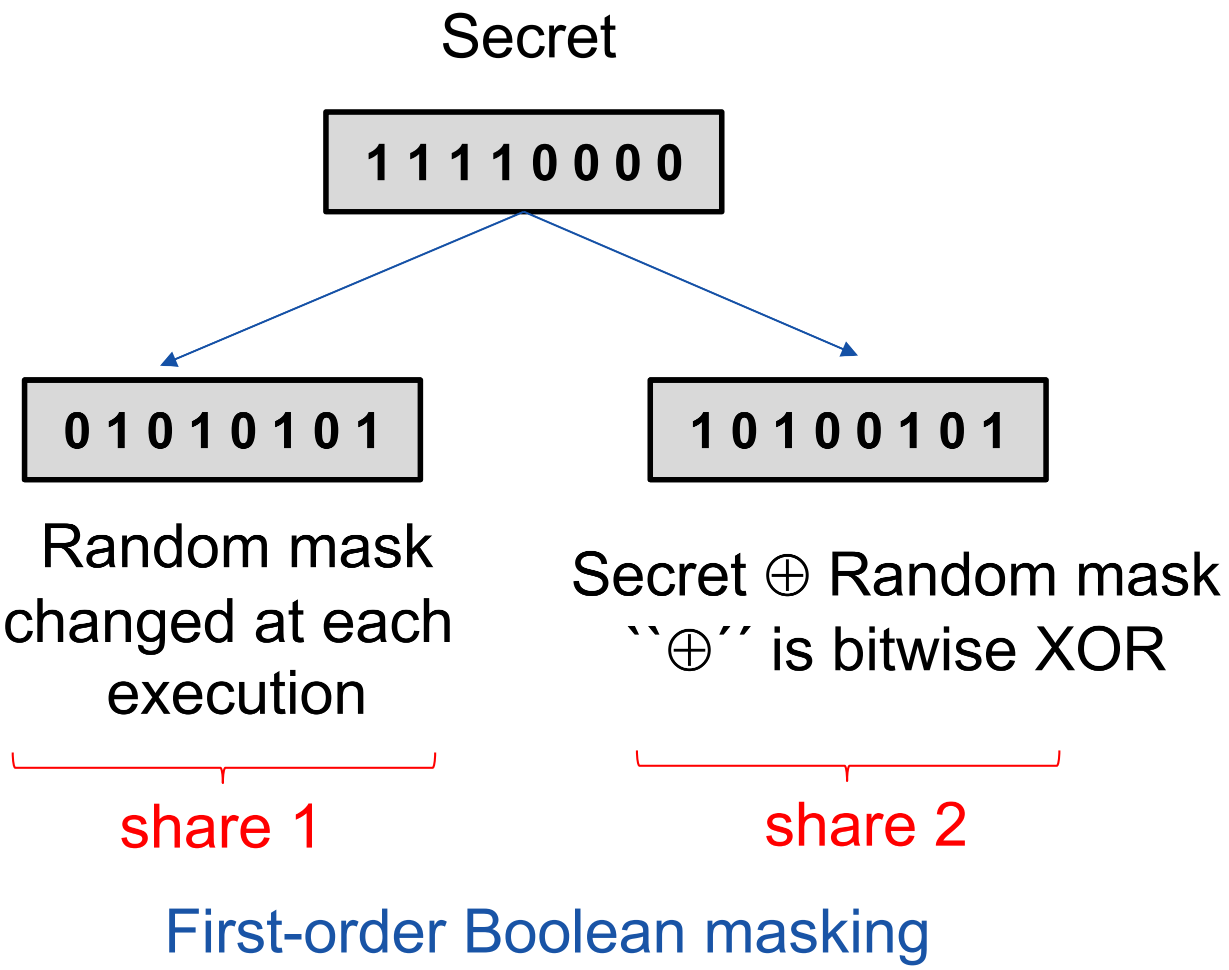


photo credit: Martin Brisfors

Masking countermeasure



NIST PQC PKE/KEM standardization process

CRYSTALS-Kyber

Selected: July 2022
Planned draft standard: 2024

Round-4 Selection

Classic McEliece (Selected by BSI, Germany)	**HQC**	**BIKE**	**SIKE** (isogeny-based, dead)

Kyber Key Encapsuation Mechanism (KEM)

- Security is based on the hardness of learning-with-errors problem in module lattices (M-LWE)

- PKE algorithms:
  - Key generation, (pk, sk) = PKE.KeyGen()
  - Encryption, c = Encrypt(pk, m, r)
  - Decryption, m = Decrypt(sk, c)

- KEM algorithms:
  - Key generation, (pk, sk) = KEM.KeyGen()
  - Encapsulation, (c, K) = Encaps(pk)
  - Decapsualtion, K = Decaps(c, sk)

▷ pk is public key
▷ sk is secret (private) key
▷ r is random coin
▷ m is message
▷ c is ciphertext, $c = (\boldsymbol{u}, v)$

▷ K is shared key

Shared key establishment protocol

Party 1

(pk, sk) = KEM.KeyGen()

Party 1 → Party 2: pk

Party 2

Encaps(pk)

$m \leftarrow \mathcal{U}(\{0,1\}^{256})$
$m = \mathcal{H}(m)$
$(\bar{K}, r)$ = $\mathcal{G}(m, \mathcal{H}(pk))$
c = Encrypt(pk, m, r)
K = KDF($\bar{K}, \mathcal{H}(c)$)

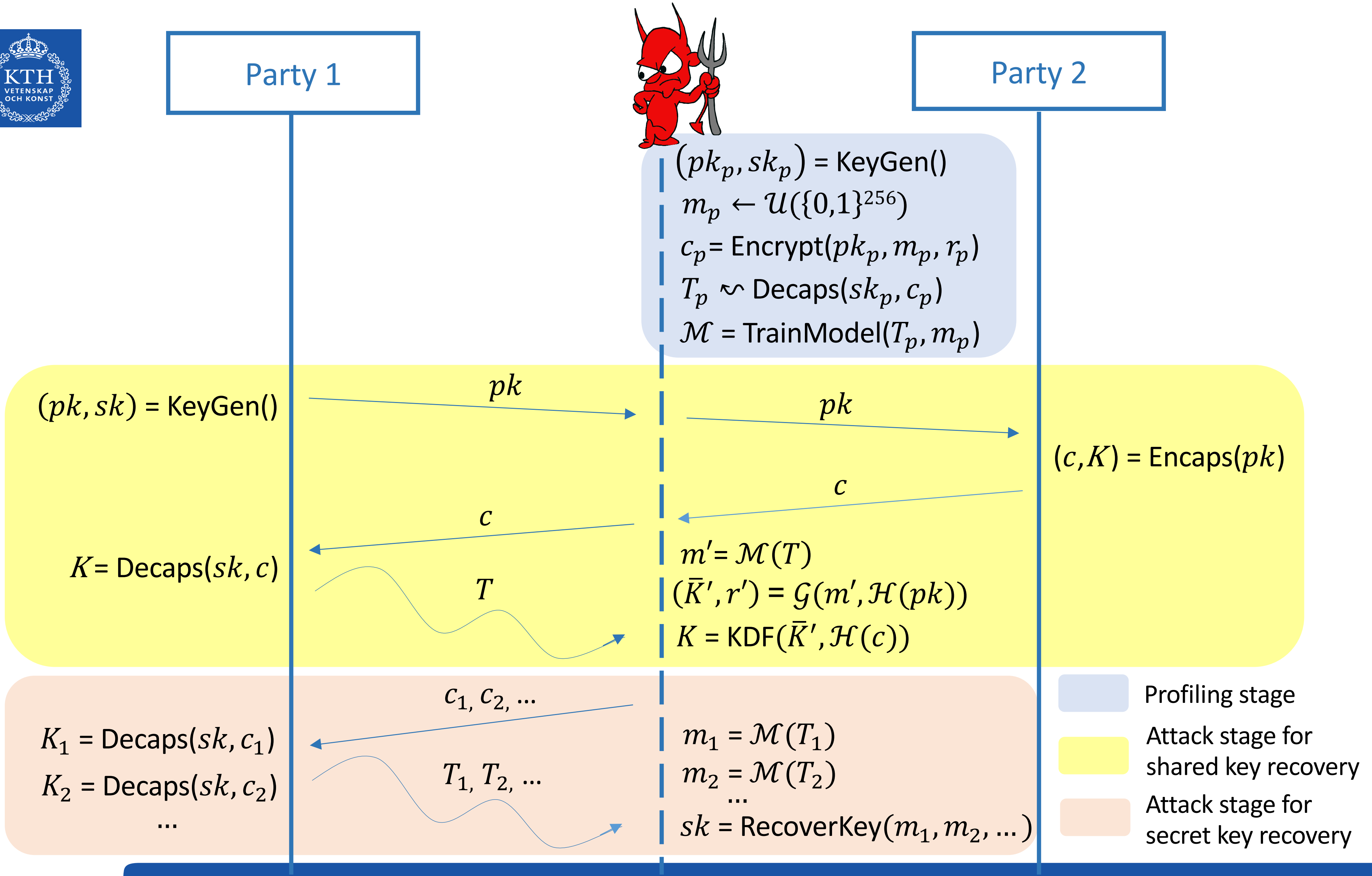
Party 2 → Party 1: c

Decaps($sk = ((\mathbf{s}, pk, \mathcal{H}(pk), z), c)$

m'= Decrypt($\mathbf{s}, c$)
$(\bar{K}', r')$ = $\mathcal{G}(m', \mathcal{H}(pk))$
c' = Encrypt(pk, m', r')
if c = c':
  K = KDF($\bar{K}', \mathcal{H}(c)$)
else
  K = KDF($z, \mathcal{H}(c)$)

Fujisaki-Okamoto transform

Assumptions :
- (pk, sk) is static
- chosen c can be decapsulated



Party 1
Party 2
(pk_p, sk_p) = KeyGen()
$m_p \leftarrow \mathcal{U}(\{0,1\}^{256})$
c_p= Encrypt(pk_p, m_p, r_p)
$T_p \backsim$ Decaps(sk_p, c_p)
$\mathcal{M}$ = TrainModel(T_p, m_p)
(pk, sk) = KeyGen()
pk
pk
(c, K) = Encaps(pk)
c
c
K= Decaps(sk, c)
T
$m'= \mathcal{M}(T)$
$(\bar{K}', r') = \mathcal{G}(m', \mathcal{H}(pk))$
K = KDF$(\bar{K}', \mathcal{H}(c))$
$c_1, c_2, \ldots$
K_1 = Decaps(sk, c_1)
K_2 = Decaps(sk, c_2)
…
$T_1, T_2, \ldots$
$m_1 = \mathcal{M}(T_1)$
$m_2 = \mathcal{M}(T_2)$
…
sk = RecoverKey$(m_1, m_2, \ldots)$
Profiling stage
Attack stage for shared key recovery
Attack stage for secret key recovery

Attack details

- Kyber C implementation (extended to higher orders):
  - Heinz, D.,Kannwischer, M.J., Land, G., Pöppelmann, T., Schwabe, P., Sprenkels: First-order masked Kyber on ARM Cortex-M4. Cryptology ePrint Archive, Report 2022/058 (2022)
  - Complied with optimization level -O3
- Attack point:
  - Re-encryption step of decapsulation
    - message encoding
- Target board:
  - ARM Cortex-M4 in CW308TSTM32F4

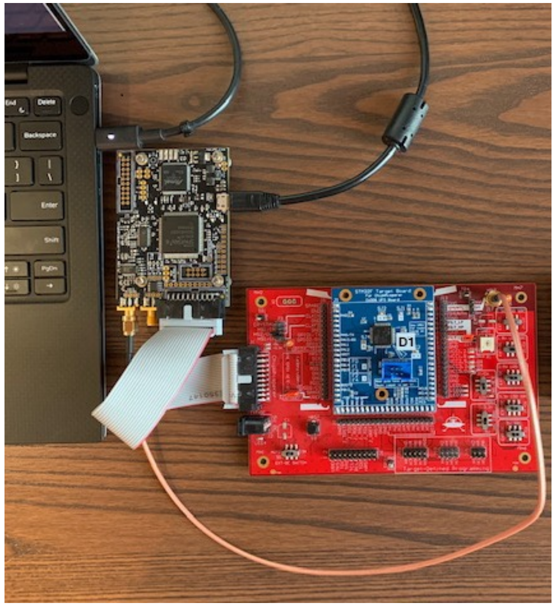


photo credit: Kalle Ngo

Encryption algorithm (simplified)

$\text{Encrypt}(pk = (seed_{\mathbf{A}}, \mathbf{b}), m, r)$

1: $\mathbf{A} \leftarrow \mathcal{U}(R_q^{k \times k}; seed_{\mathbf{A}})$
2: $\mathbf{r} \leftarrow \mathcal{B}_{\eta_1}(R_q^{k \times 1}; r)$
3: $\mathbf{e}' \leftarrow \mathcal{B}_{\eta_2}(R_q^{k \times 1}; r)$
4: $e'' \leftarrow \mathcal{B}_{\eta_2}(R_q^{1 \times 1}; r)$
5: $\mathbf{u} = \left\lfloor (\mathbf{r} \cdot \mathbf{A} + \mathbf{e}') \cdot 2^{d_u} / q \right\rceil$
6: $v = \left\lfloor (\mathbf{r} \cdot \mathbf{b} + e'' + \text{encode}(m)) \cdot 2^{d_v} / q \right\rceil$
7: **return** $c = (\mathbf{u}, v)$

Message encoding:

Converts an array of bytes representing a message m into a polynomial with coefficients $\lfloor q/2 \rceil \cdot m[j]$, where $m[j]$ is j^{th} bit of m

Non-masked message encoding in Kyber implementation of Kannwischer et al.

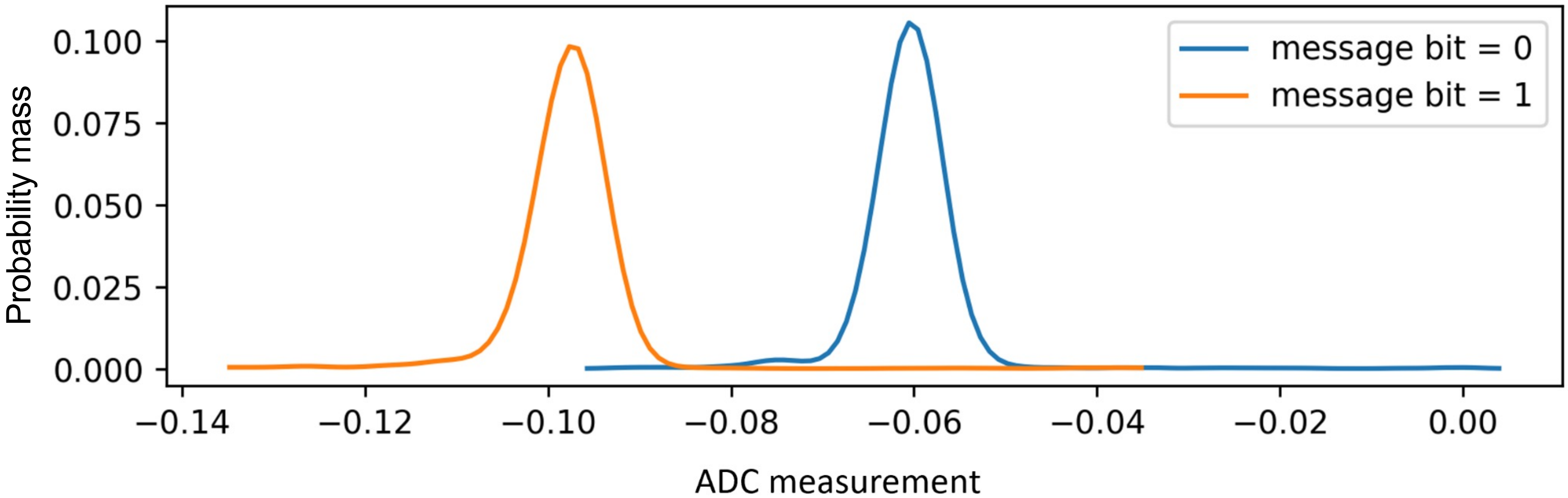
```
function POLY_FROMMSG(poly *r, unsigned char msg[32])
    uint16 mask
    for (int i=0; i < 32; i++) do
        for (int j=0; j < 8; j++) do
            mask = -((msg[i] >> j) &) 1 )          /* bit extraction */
            r.coeff[8*i+j] = mask & ((KYBER_Q + 1)/2)
        end for
    end for
end function
```

Mask takes values 0x0000 or 0xFFFF

Large difference in Hamming weight ⇒ easy to distinguish

First described by Amiet et al. for NewHope KEM, ICPQC'2020

Distributions of power consumption for message bits



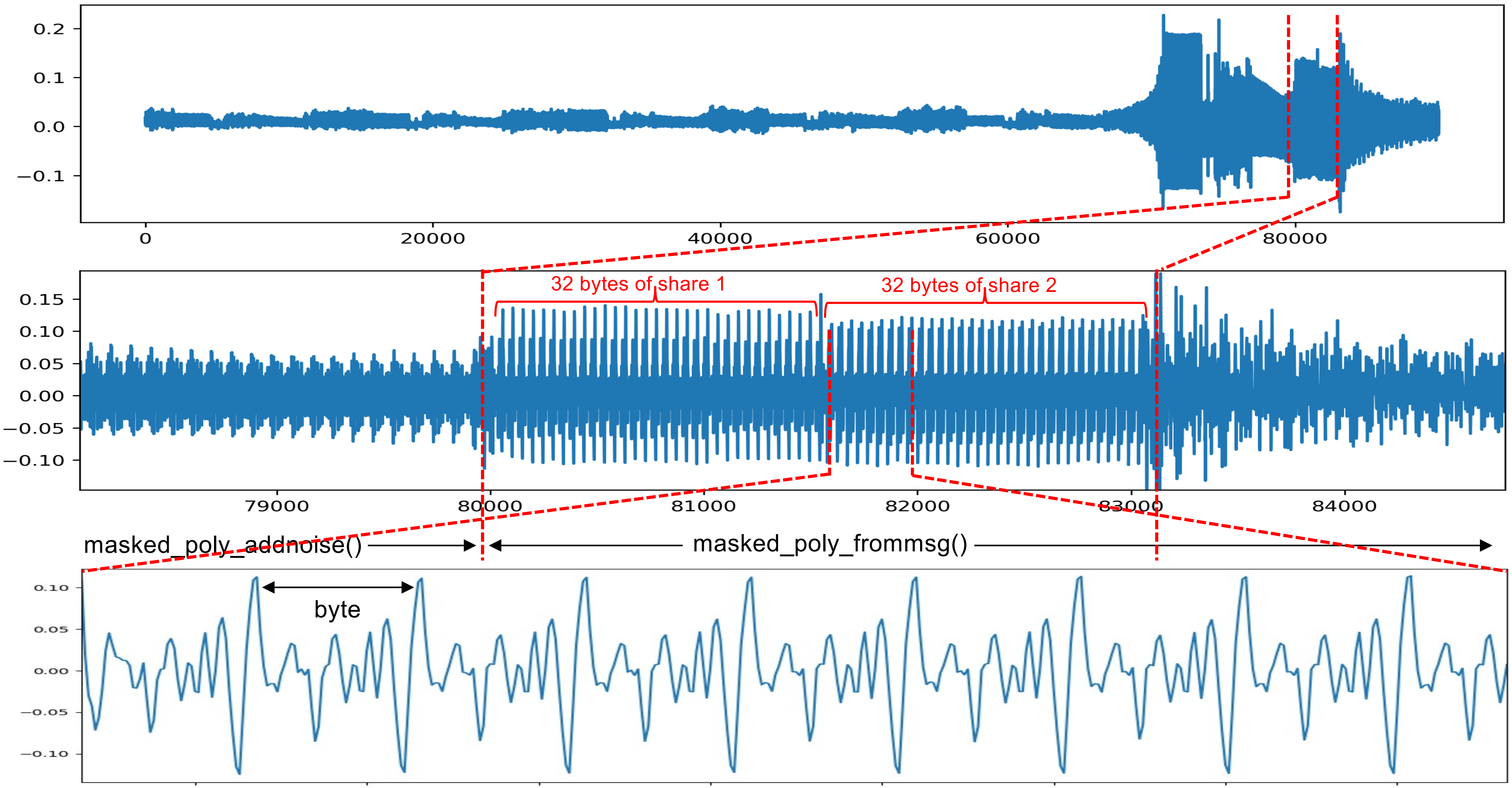
Non-overlapping distributions $\Rightarrow$ easy to distinguish

Masked message encoding in Kyber implementation on Heinz et al.

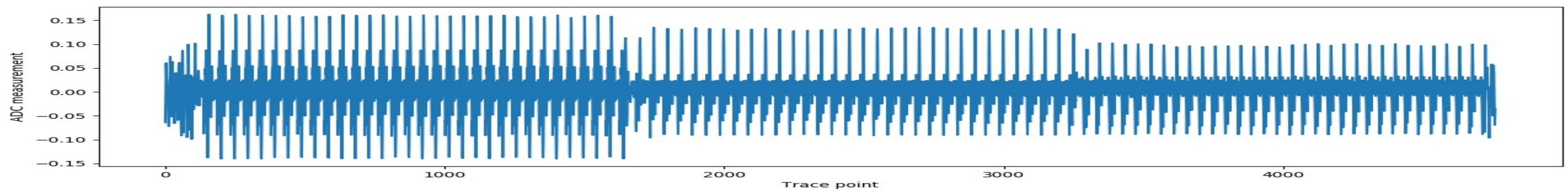
```
void masked_poly_frommsg(uint16 poly[2][256],
uint8 msg[2][32])

for (i = 0; i < 32; i++) do
    for (j = 0; j < 8; j++) do
        mask = -((msg[0][i] » j) & 1);    /* Boolean share 0 bit extraction */
        poly[0][8*i+j] += (mask&((KYBER_Q+1)/2));
    end for
end for
for (i = 0; i < 32; i++) do
    for (j = 0; j < 8; j++) do
        mask = -((msg[1][i] » j) & 1);    /* Boolean share 1 bit extraction */
        poly[1][8*i+j] += (mask&((KYBER_Q+1)/2));
    end for
end for
...
```

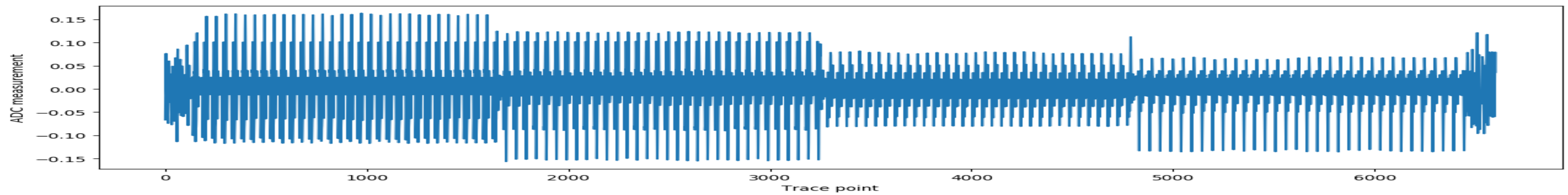
Segment of power trace of re-encryption in Kyber implementation on Heinz et al.



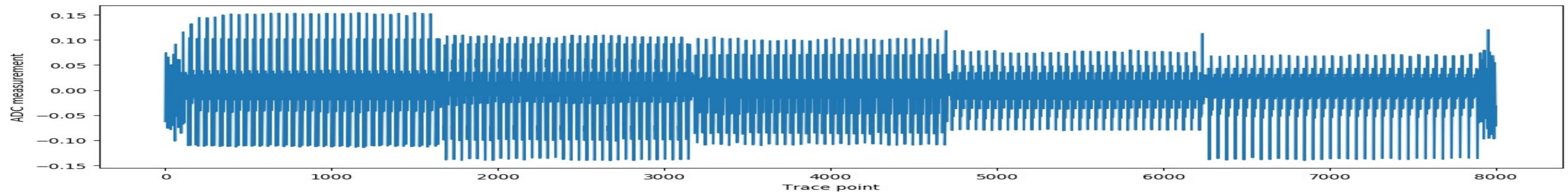
More shares ⇒ more 32-byte blocks



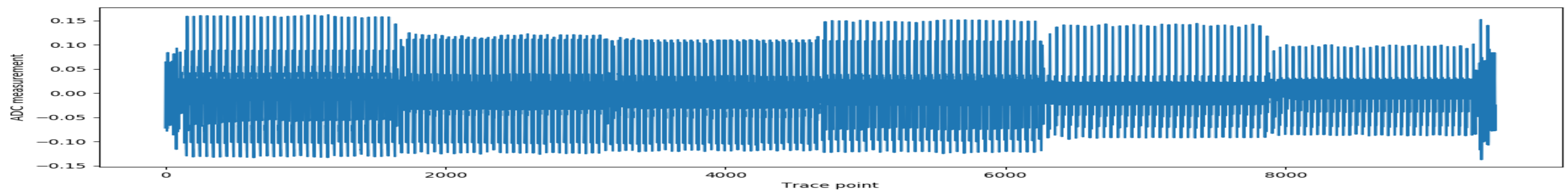
3



4



5



6

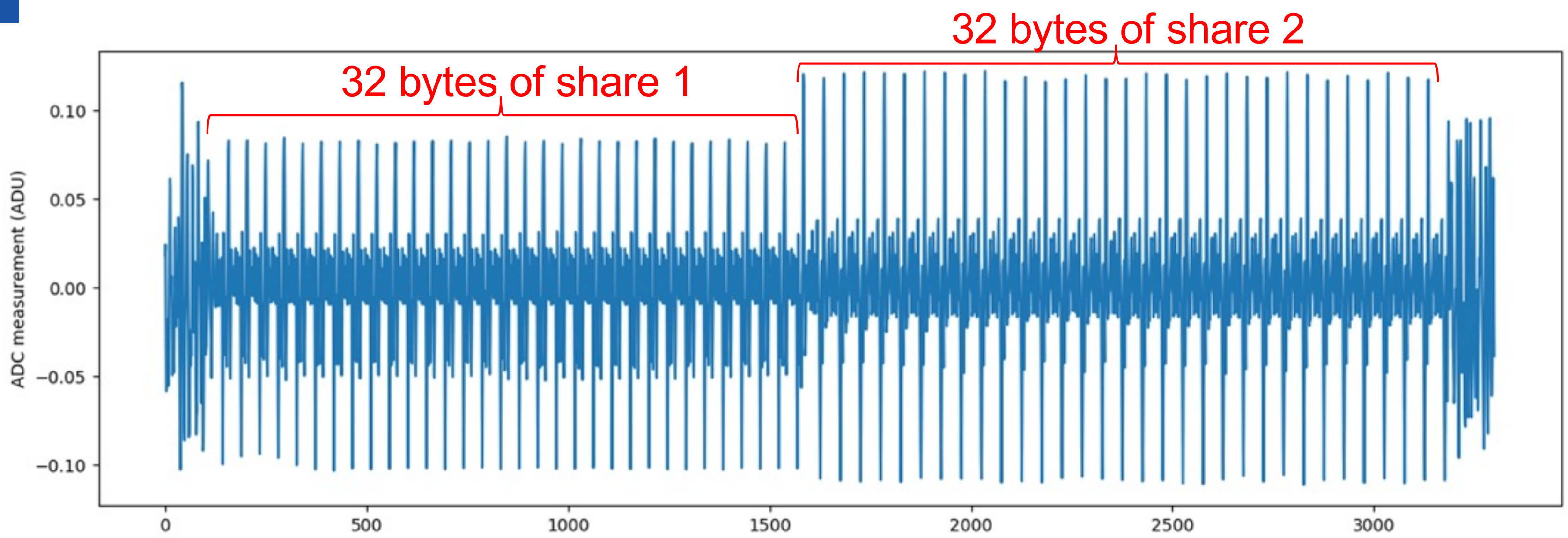
MLP architecture for message bits recovery from an ω-order masked implementation

Profiling strategy:

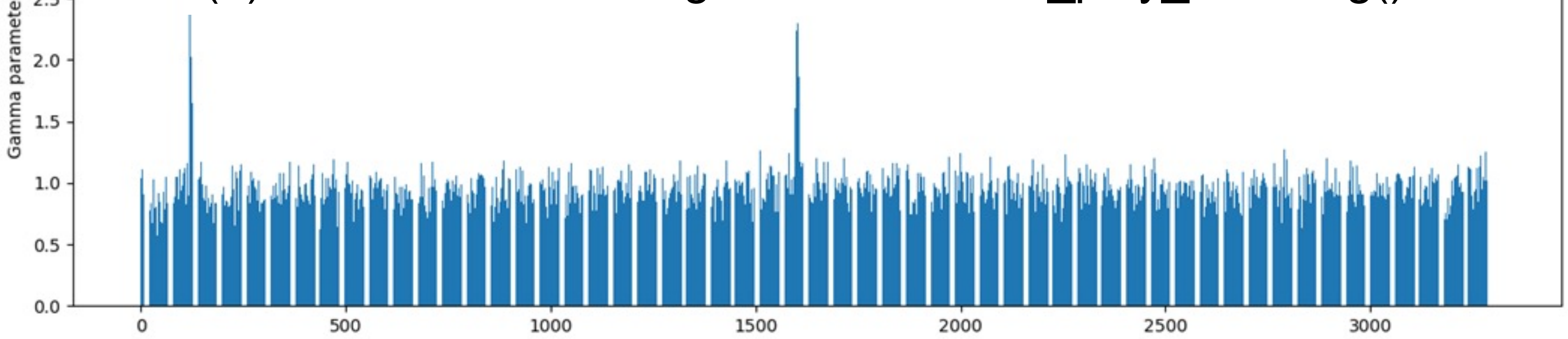
- For each $\omega \in \{1,\ldots,5\}$, we use 30K training set cut-and-joined on 32 bytes, 30K×32 = 960K
- Message bit values are used as labels for traces

Layer type	Output shape	*ω = 1*
Input	$32(\omega+1)$	
Batch Normalization 1	$32(\omega+1)$	
Dense 1	$32(\omega+1)$	*64*
Batch Normalization 2	$32(\omega+1)$	
ReLU	$32(\omega+1)$	
Dense 2	$2^{\omega+4}$	*32*
Batch Normalization 3	$2^{\omega+4}$	
ReLU	$2^{\omega+4}$	
Dense 3	$2^{\omega+3}$	*16*
Batch Normalization 4	$2^{\omega+3}$	
ReLU	$2^{\omega+3}$	
Dense 4	1	
Softmax	1	

How to decide where to cut?

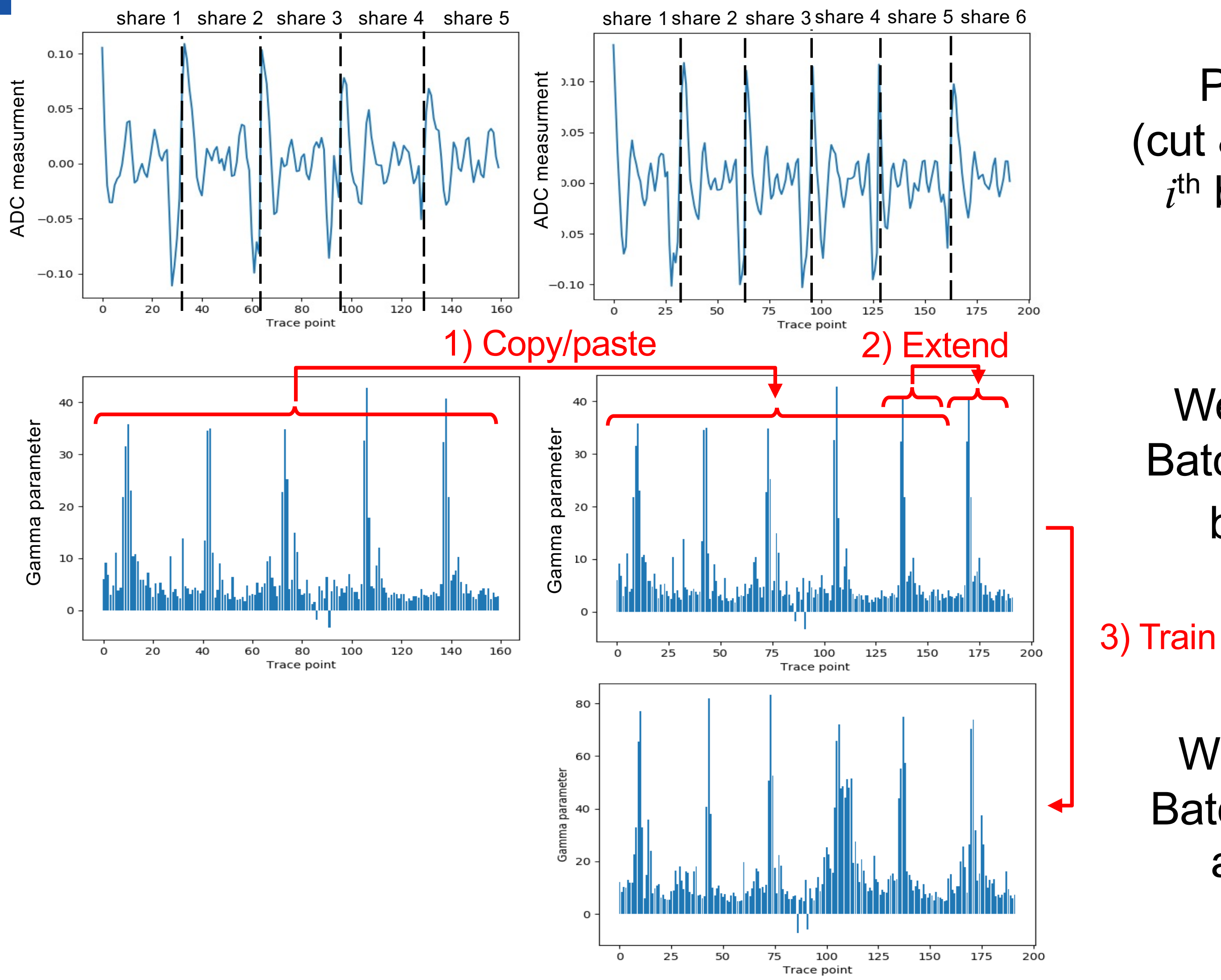


(a) Power trace of a segment of masked_poly_frommsg()



(b) Weights of MLP BatchNormalization1 layer after training for m[0] bit

Copy-paste method



Power traces
(cut & concatenated
i^{th} bits of shares)

Weights of MLP
BatchNorm.1 layer
before training

Weights of MLP
BatchNorm.1 layer
after training

Attack results for the first-order masking

Attack type	Mean empirical probability to recover i^{th} message bit								Avg.
	0	1	2	3	4	5	6	7	
Single-trace	0.9992	0.9989	0.9953	0.9841	0.9876	0.9835	0.9393	0.9067	0.9743
With 4 rotations	0.9994	0.9991	0.9993	0.9990	0.9988	0.9885	0.9993	0.9992	0.9991

Four-trace attack results, ω-order masking (captured with 4 negacyclic rotations)

ω	Mean empirical probability to recover i^{th} message bit								Avg.
	0	1	2	3	4	5	6	7	
1	0.9994	0.9991	0.9993	0.9990	0.9988	0.9885	0.9993	0.9992	0.9991
2	0.9983	0.9979	0.9986	0.9980	0.9992	0.9982	0.9985	0.9976	0.9983
3	0.9978	0.9958	0.9971	0.9951	0.9971	0.9945	0.9979	0.9958	0.9964
4	0.9947	0.9775	0.9951	0.9764	0.9947	0.9763	0.9947	0.9771	0.9858
5	0.9924	0.9682	0.9918	0.9661	0.9923	0.9677	0.9937	0.9673	0.9799

ω	1	2	3	4	5
p_{mesage}	0.7887	0.6857	0.3964	0.0259	0.0056

20-trace attack results for 5-order masking (with 4 negacyclic rotations and 5 repetitions)

ω	Mean empirical probability to recover i^{th} message bit								Avg.
	0	1	2	3	4	5	6	7	
5	1.0000	0.9987	1.0000	0.9989	1.0000	0.9992	1.0000	0.9988	0.9995

ω	5
p_{mesage}	0.8709

Since ranom masks are updated at each execution, errors in repeated measurments are more independent than in the non-masked case

Summary

- Some higher-order masked software implementations of Kyber can be broken by power analysis
- Cyclic rotations are useful for the attacker

Future work

- Design stronger, DL-resistant countermeasures for software implementations of PQC algorithms
- Analyze hardware implementations of PQC algorithms
  - Ongoing analysis of the masked FPGA implementation of Kyber by Kamucheka et al. presented at the NIST 4th PQC Standardization Conference, Nov. 2022
  - Ongoing analysis of our own protected FPGA implementation of Kyber built on the top of Xing et al. implementation presented at TCHES'2021

Thank you!