How We Broke a Fifth-Order Masked Kyber Implementation by Copy-Paste

Elena Dubrova, Kalle Ngo, Joel Gärtner
School of Electrical Engineering and Computer Science
KTH Royal Institute of Technology
Stockholm, Sweden

Outline

- Background
  - Side-channel analysis
  - Masking & shuffling countermeasures
  - IND-CCA2 secure Kyber KEM
- Side-channel attack on a higher-order masked Kyber implementation
  - Profiling strategy
  - Copy-paste method
  - Experimental resuts
- Summary & future work

How side-channel attacks work?

- 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

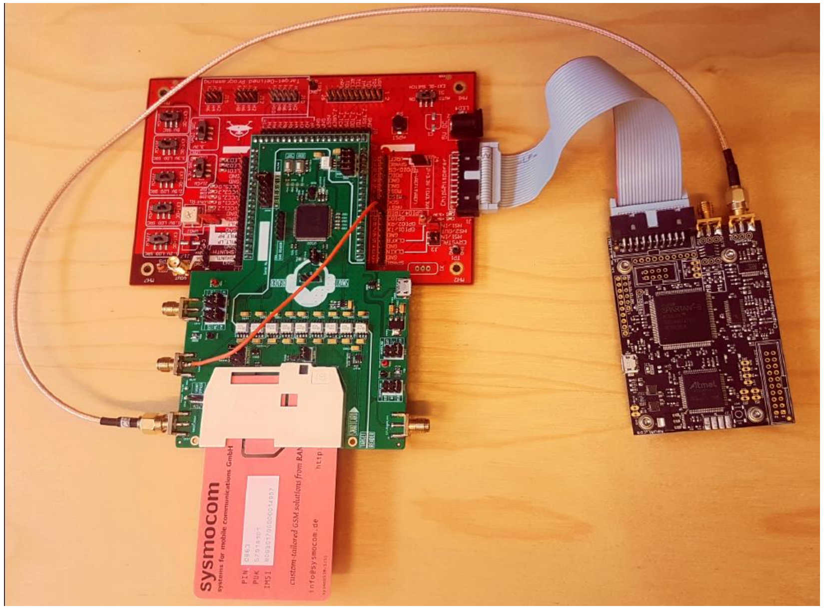
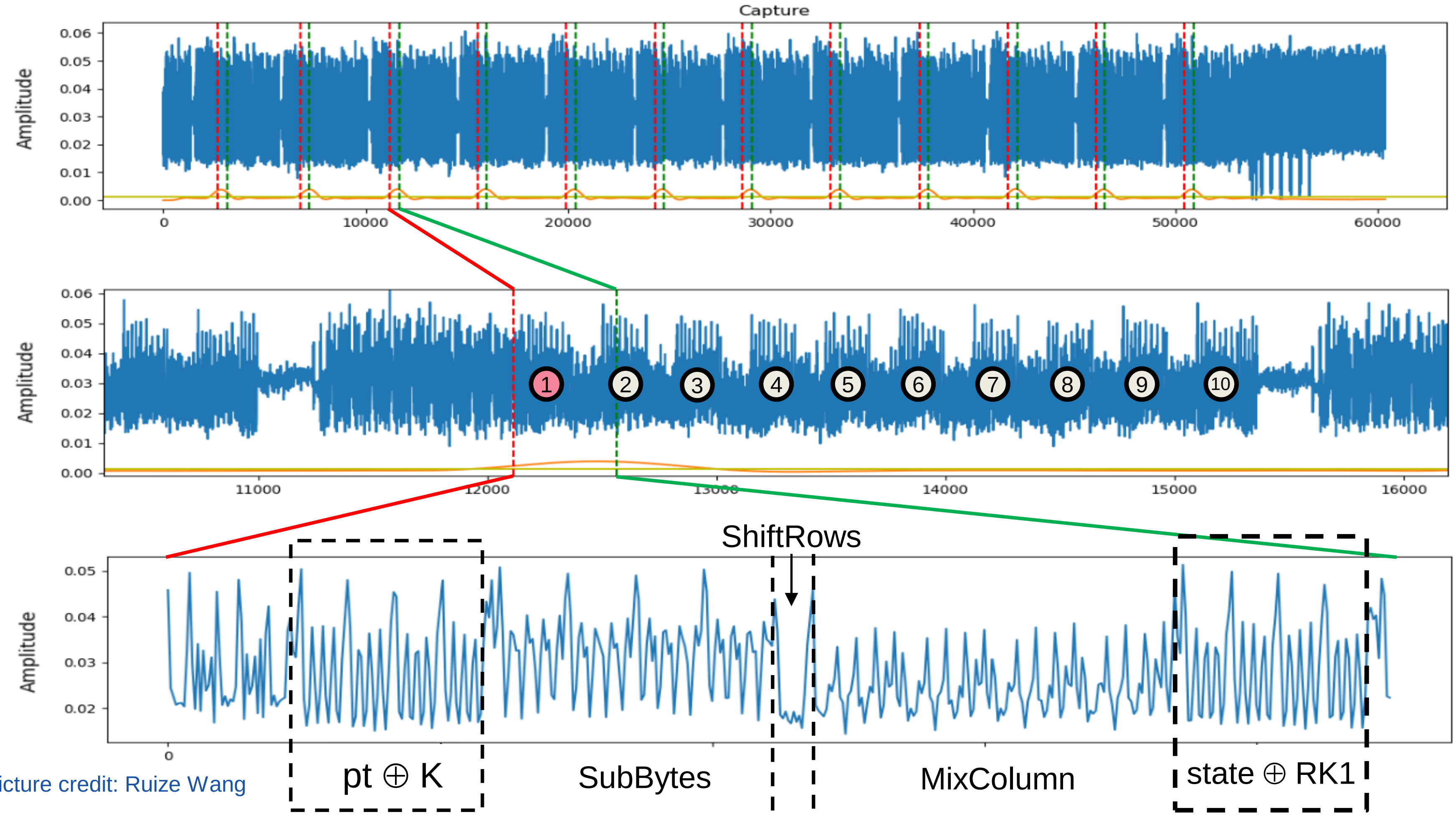


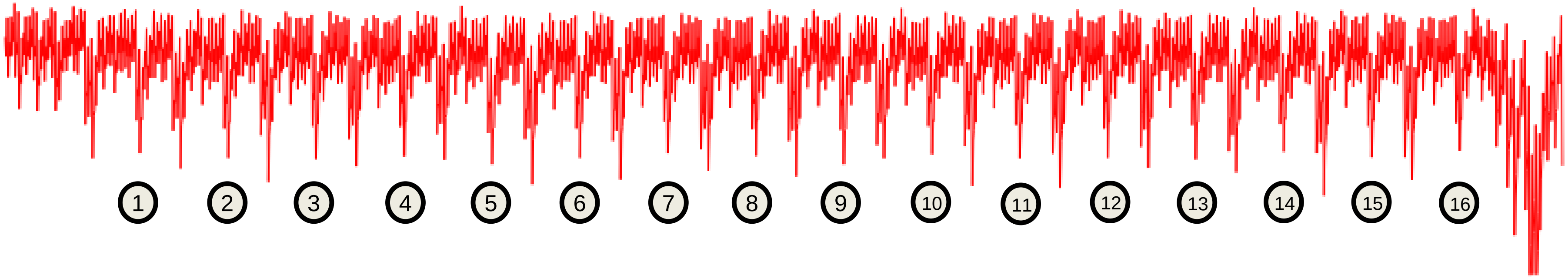
photo credit: Martin Brisfors

Recognizing operations: AES-128 in 32-bit MCU



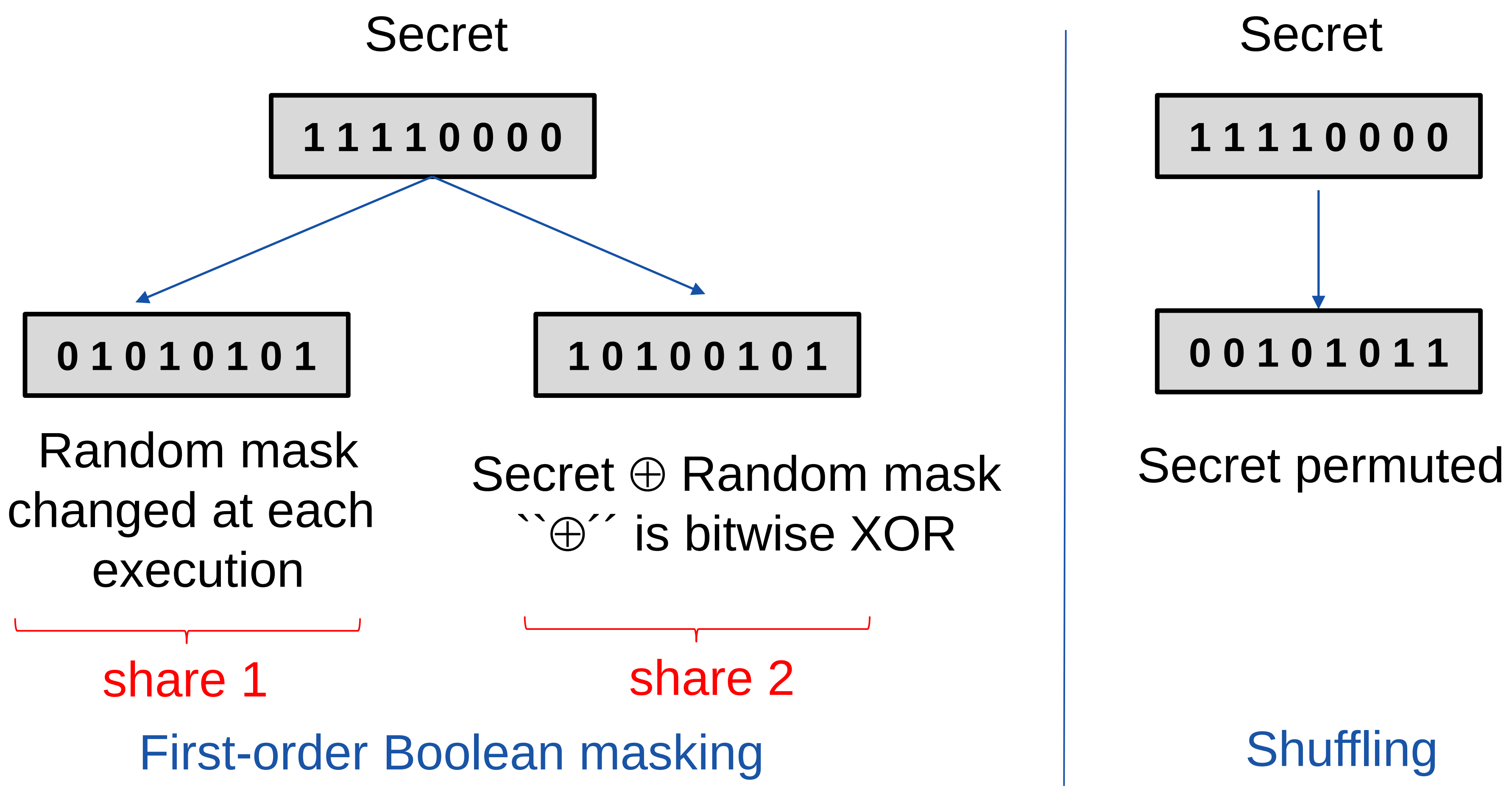
picture credit: Ruize Wang

Recognizing data: AES-128 in 8-bit MCU



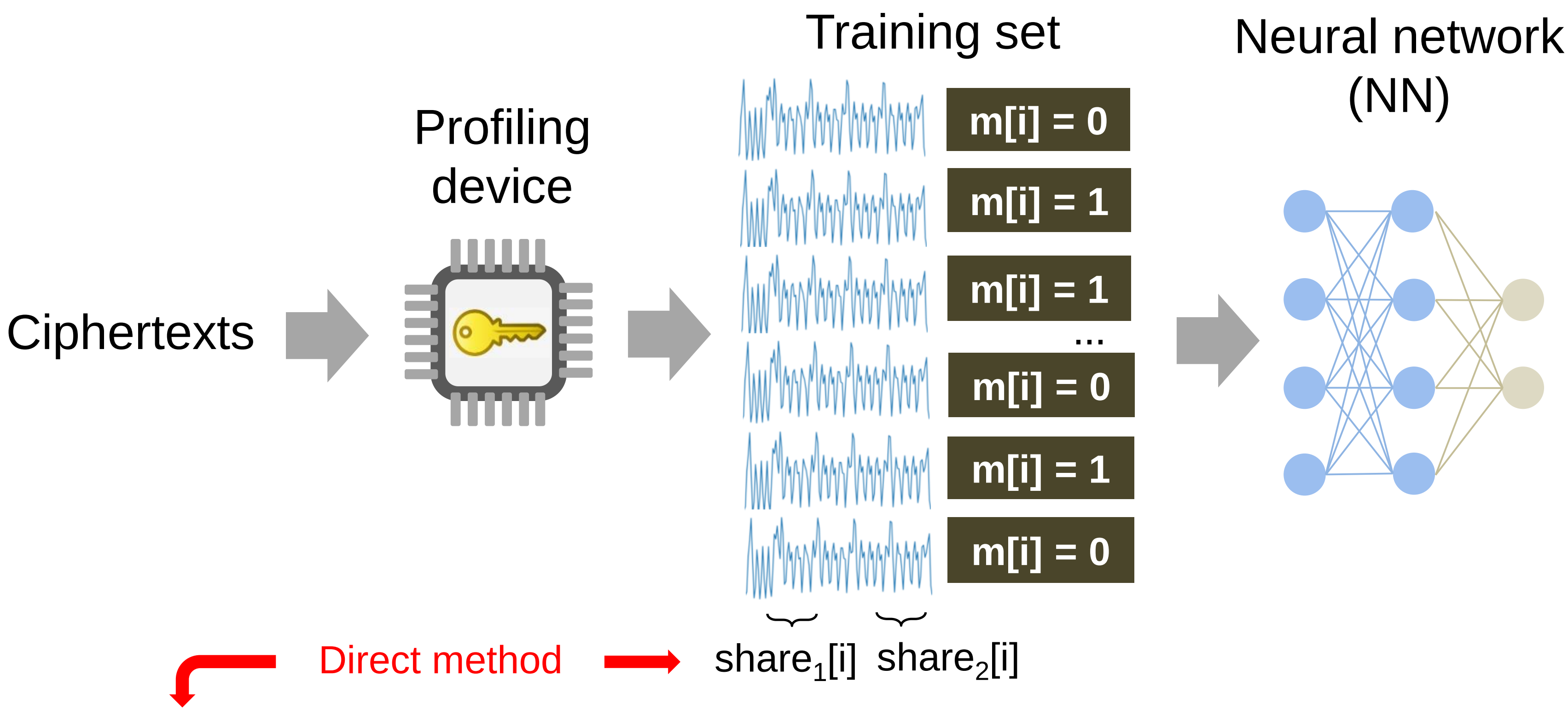
16 executions of SubBytes

Masking and shuffling countermeasures



Deep learning-based side-channel analysis

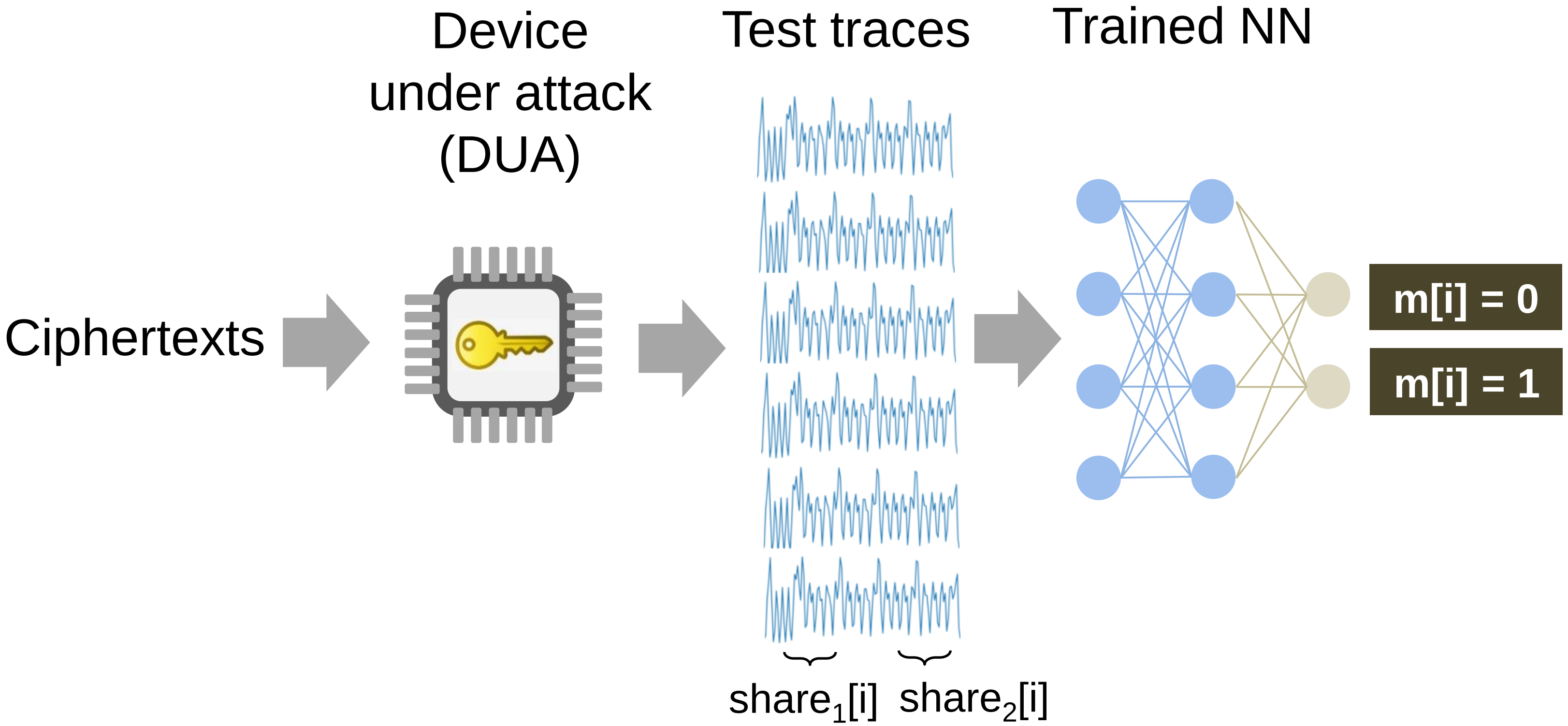
Profiling stage: Train a NN using traces from profiling device(s)



Ngo, K., Dubrova, E., Guo, Q., Johansson, T., A side-channel attack on a masked IND-CCA secure Saber KEM implementation, TCHES'2021

Deep learning-based side-channel analysis, cont.

Attack stage: Use the trained NN to classify traces from DUA



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)

- A version of Fujisaki-Okamoto transform is used to create an IND-CCA2 secure KEM from an IND-CPA secure PKE

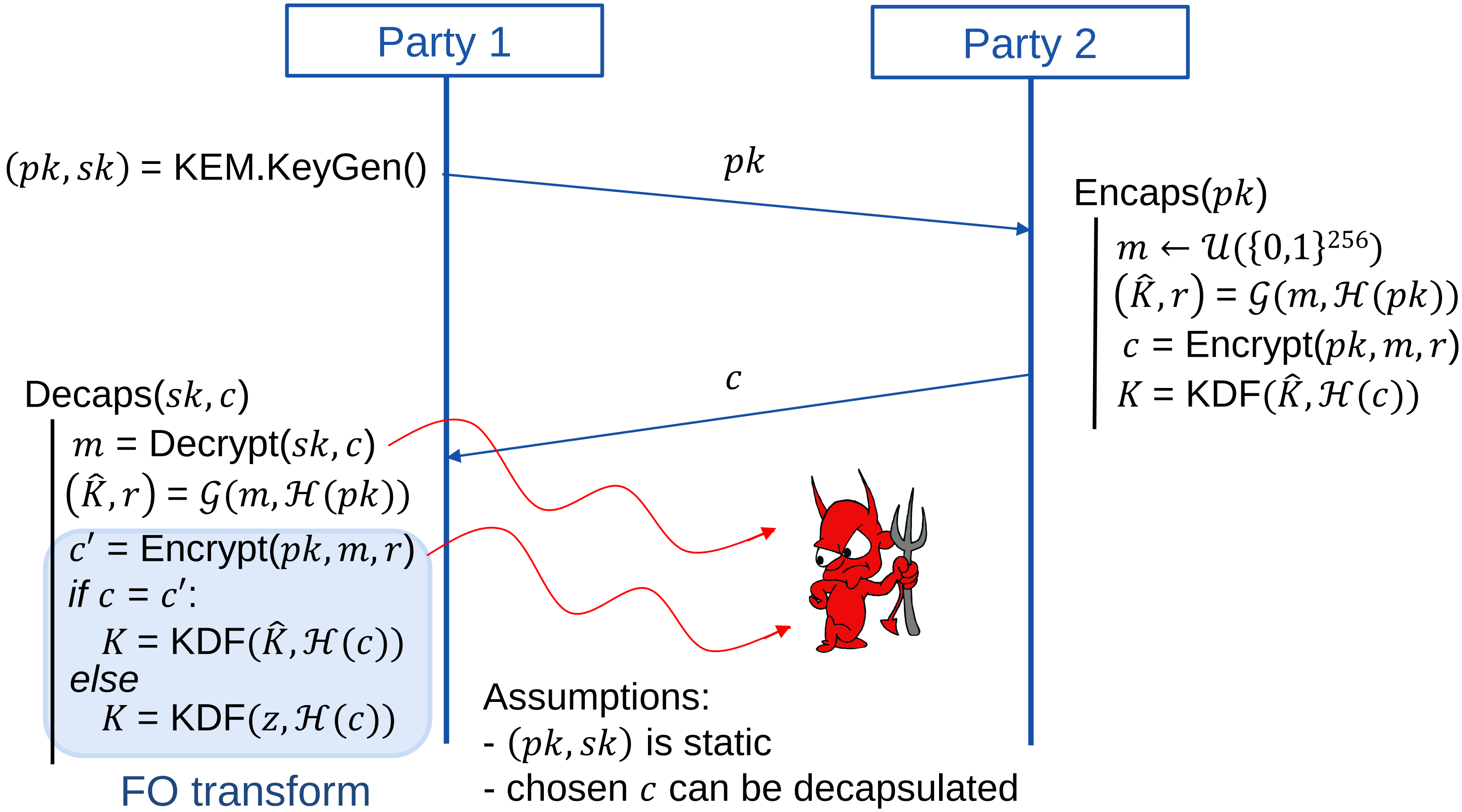
- PKE algorithms:
  - Key generation, (pk, sk) = PKE.KeyGen()
  - Encryption, c = Encrypt(pk, m, r)
  - Decryption, m = Decrypt(sk, c)

- 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

▷ K is shared key

Shared key estabilishment protocol



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

Attack point: poly_frommsg() of re-encryption

```
function POLY_FROMMSG(poly *r, unsigned char msg[32])
    uint16 mask
    for (int i=1; i < 32; i++) do
        for (int j=0; j < 8; j++) do
            mask = -((msg[i] >> j) & 1 )
            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. in an attack on NewHope KEM, Int. Conf. on PQC, 2020

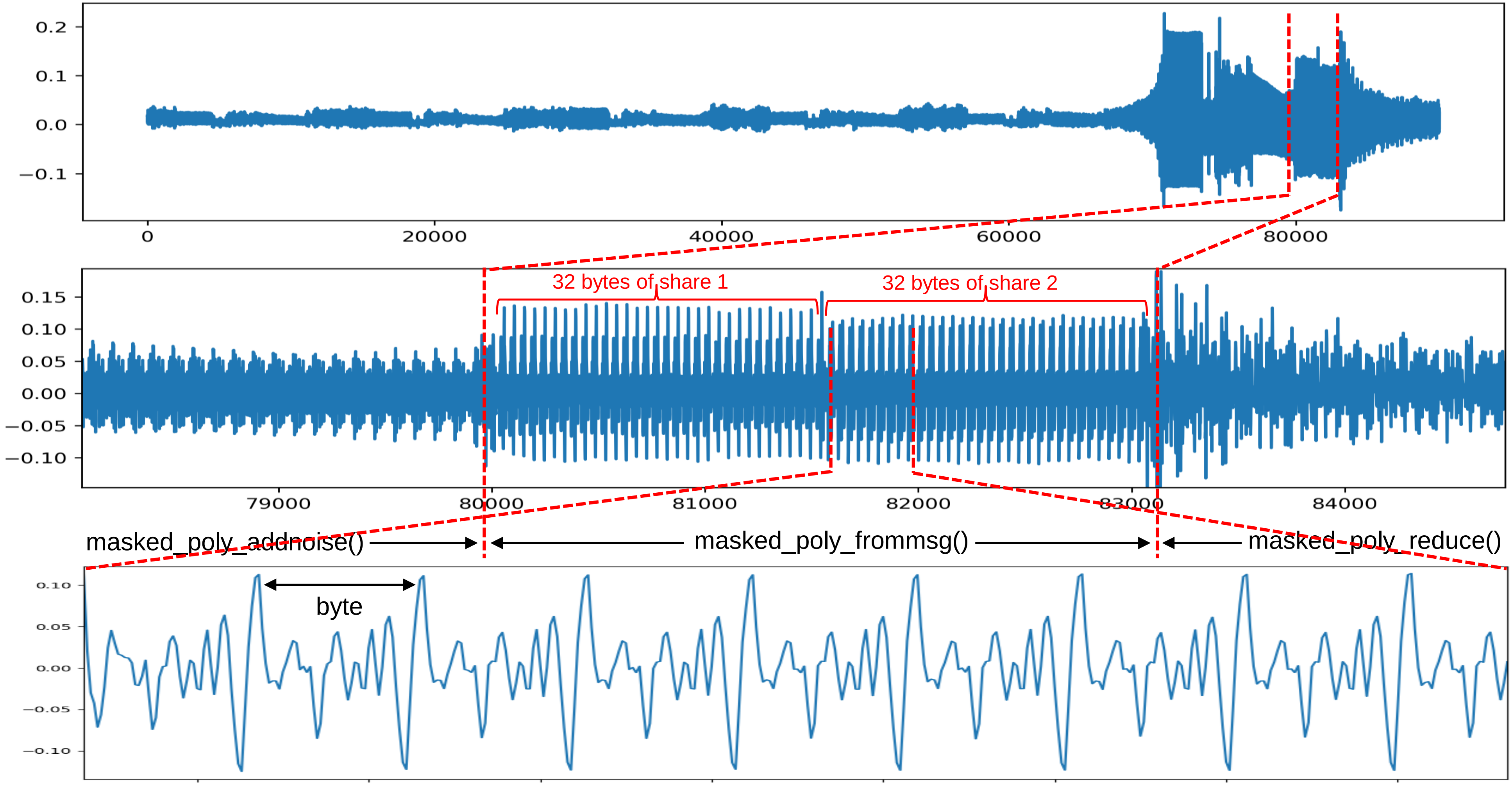
Implementation of poly_frommsg() in masked implementation of Kyber

Heinz, D., Kannwischer, M.J., Land, G., Pöppelmann, T., Schwabe, P., Sprenkels, D., First-order masked Kyber on ARM Cortex-M4, Cryptology ePrint Archive, 2022/058

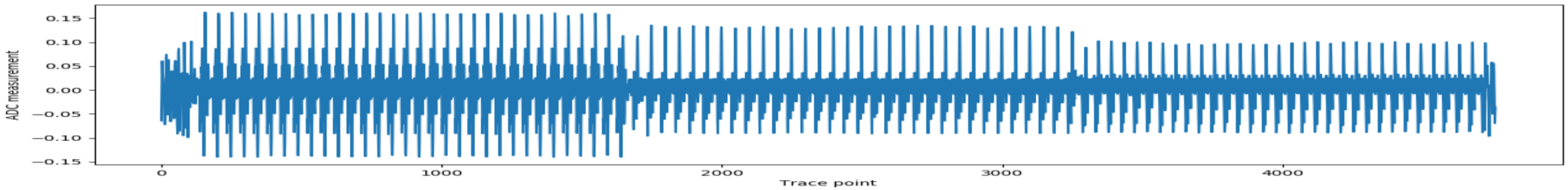
```
void masked_poly_frommsg(uint16 poly[2][256],
uint8 msg[2][32])

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

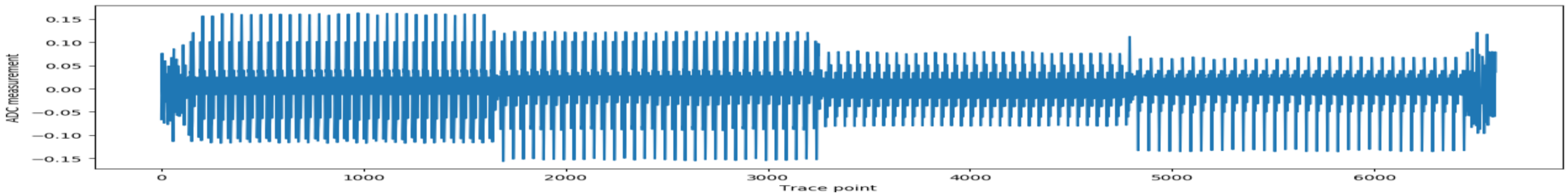
Power trace of re-encryption of Kyber768 implementation in ARM Cortex-M4



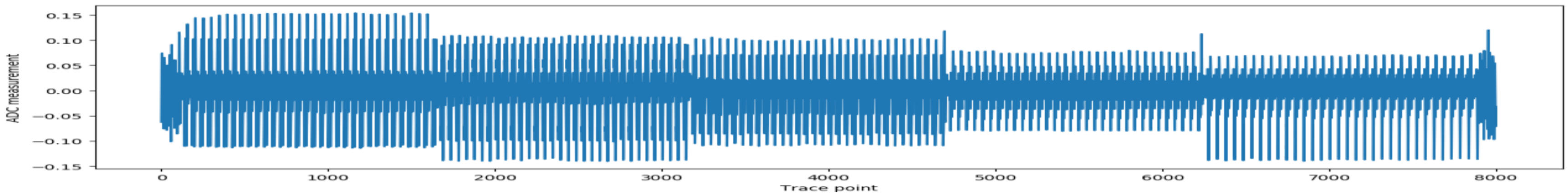
More shares ⇒ more 32-byte blocks



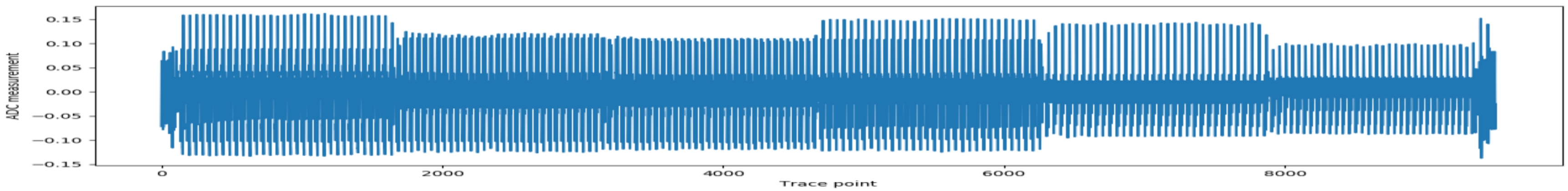
3



4



5



6

What we knew before the start?

- First-order masked implementation of Saber KEM by Beirendonck et al. (JETC’21) can be broken by the direct method (Ngo et al. TCHES’2021, Paulsrud KTH MSc thesis, 2022)
- Second- and third-order masked implementation of Saber KEM by Kundu et al. (https://eprint.iacr.org/2022/389) can be broken by the direct method (Ngo et al. https://eprint.iacr.org/2022/919)
- Adding shuffling to the first-order masked implementations of Saber by Beirendonck et al. (JETC’21) and Kyber by Heinz [1] does not prevent the attack (Ngo el al. ASHES’21, Backlund et al. https://eprint.iacr.org/2022/1692)
    - A shuffled implementation must be profiled on another device

What we knew before the start?, cont.

- DL can learn from very noisy traces (Wang et al, ICISC’2022)
  - Attack on Kyber using amplitude-modulated EM emanations from an nRF52832 SoC
    - ARM Cortex-M4
    - Multi-protocol 2.4GHz radio

Question:

How many shares can the direct method handle?

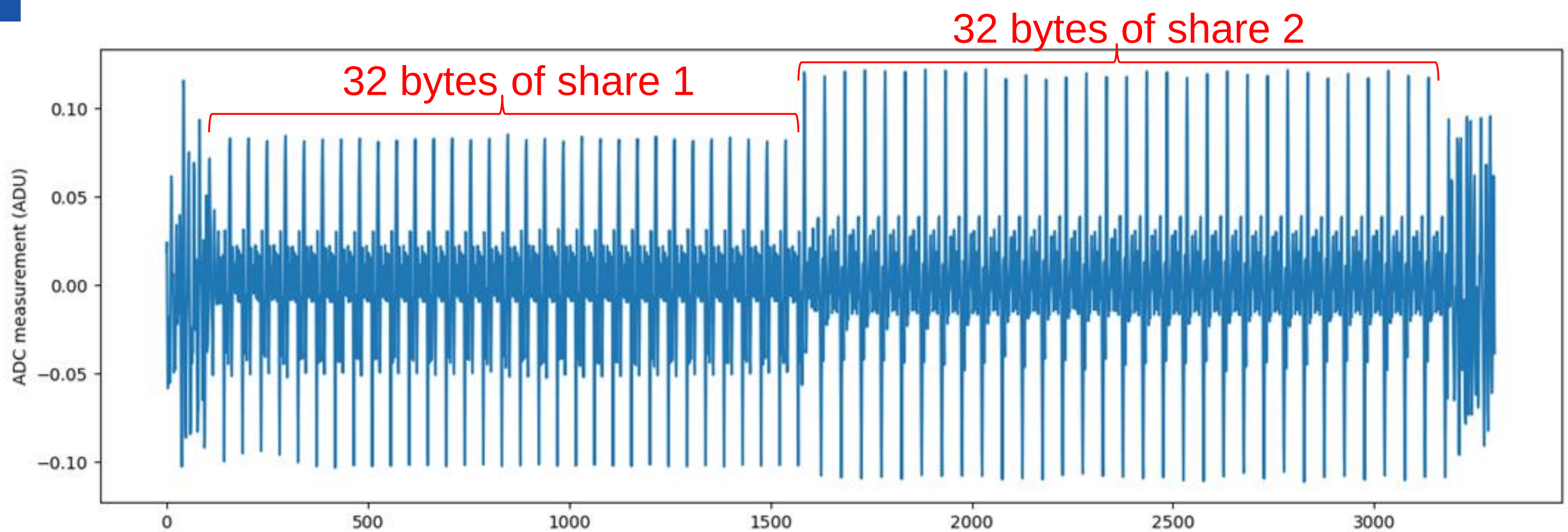
MLP architecture for message bits recovery from an ω-order masked implementation

Profiling strategy:

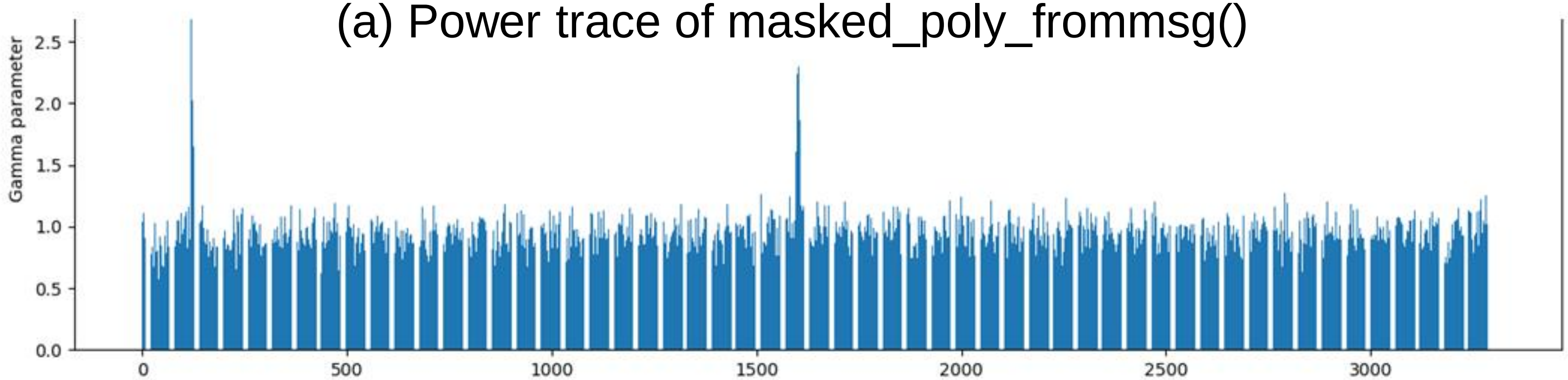
- For each $\omega \in \{1,\ldots,5\}$, we use 30K training set cut-and-joined on 32 bytes, 30K×32 = 960K
  - Captured from DUA
  - CW308T-STM32F4 board
  - C implementation of Kyber is compiled with –O3
- Message bit values are used as labels for traces

Layer type	Output shape	$\omega = 1$
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	$2^{\omega+3}$	
Output	1	
Softmax	1	

How to decide where to cut?

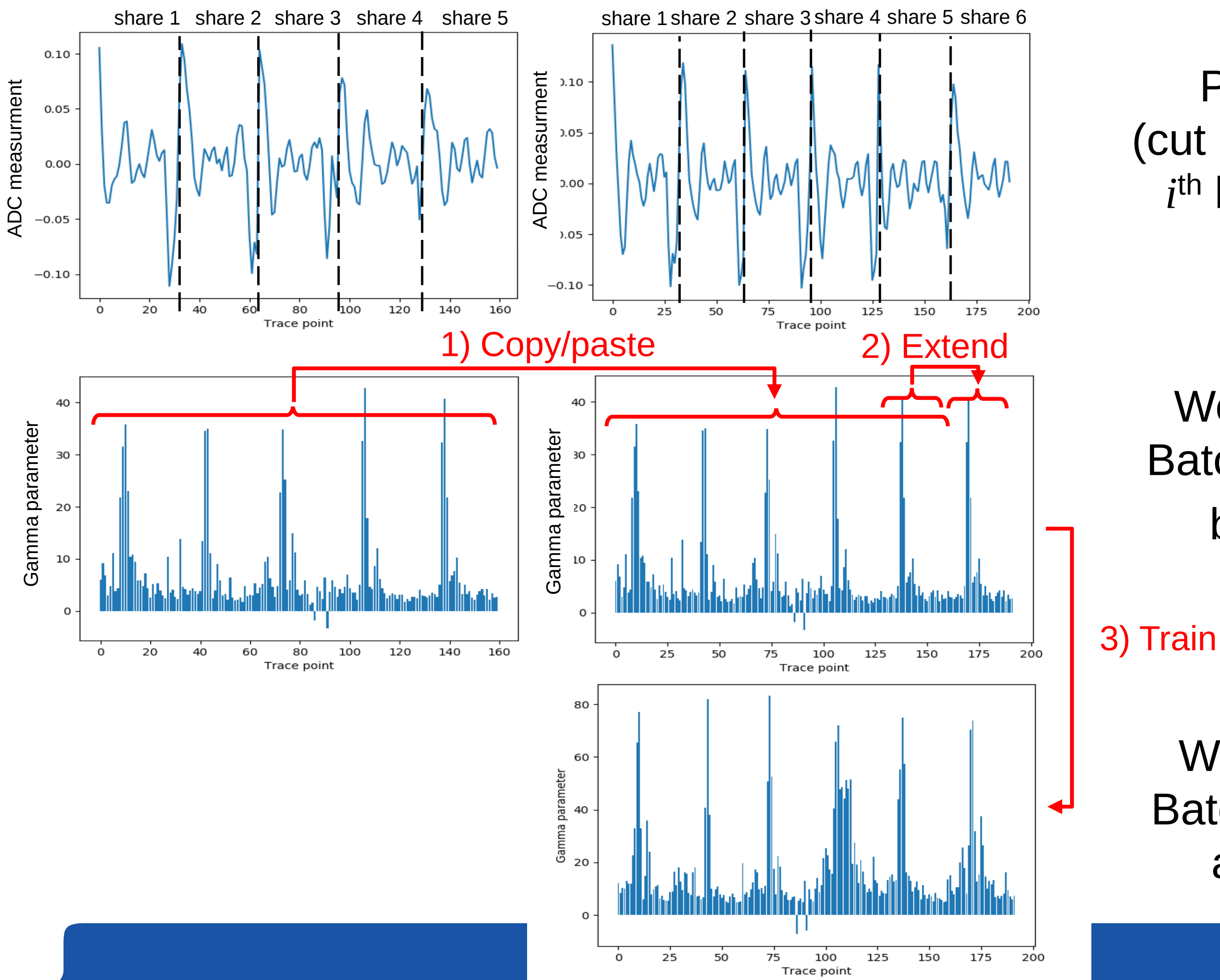


(a) Power trace of masked_poly_frommsg()



(b) Weights of MLP BatchNorm.1 layer after training (1st message 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

Messages of some LWE/LWR-based PKE/KEMs can be cyclically rotated by manipulating the ciphertext

Ravi, P., Bhasin, S., Roy, S., Chattopadhyay, A., On exploiting message leakage in (few) NIST PQC candidates for practical message recovery and key recovery attacks, https://eprint.iacr.org/2020/1559.pdf

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 less dependent

Summary

- Copy-paste method enables message recovery from higher-order masked impementations using the direct method
  - helps DL start in a right place
- Cyclic rotations are useful
- Repetititons are useful
  (for the attacker)

source: Hedgehog in the Fog, Soyuzmultfilm , 1975

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!