Side-channel analysis countermeasures for lattice-based cryptography

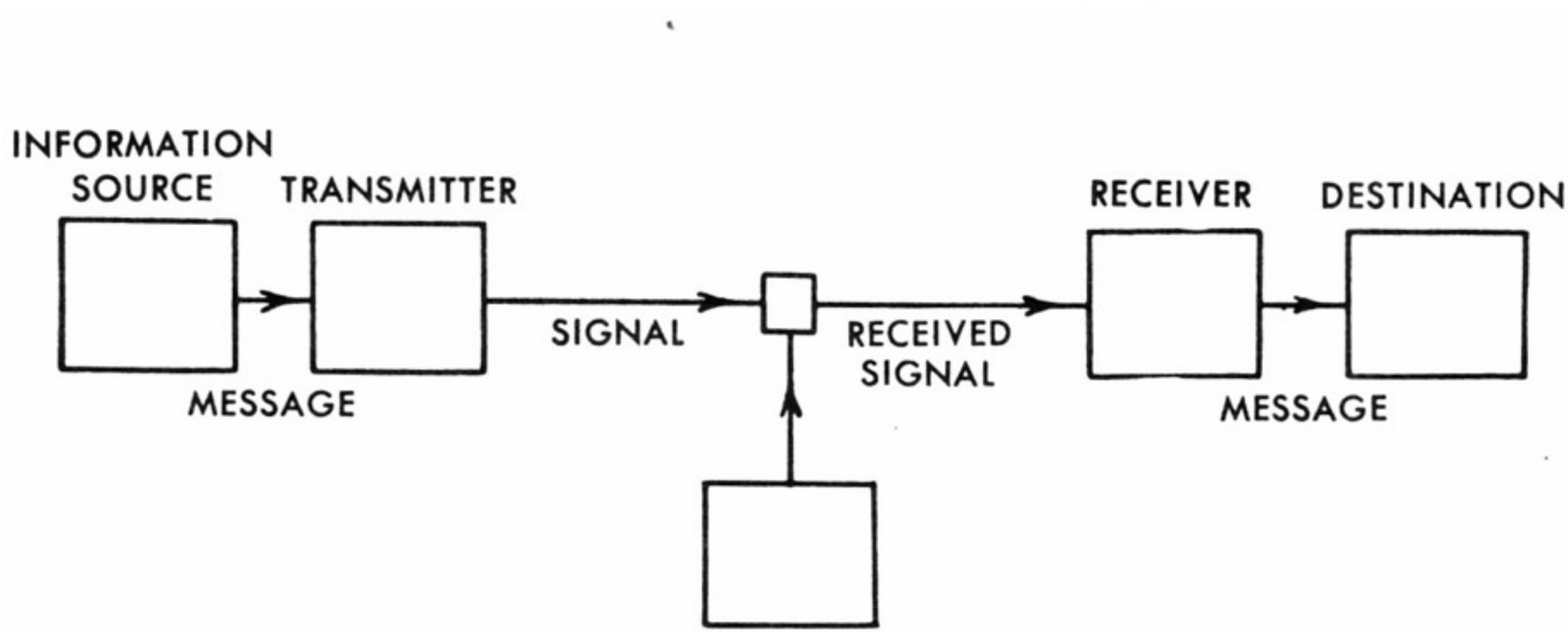
Oscar Reparaz

PQCRYPTO mini-workshop
2016.06.28, Utrecht (NL)

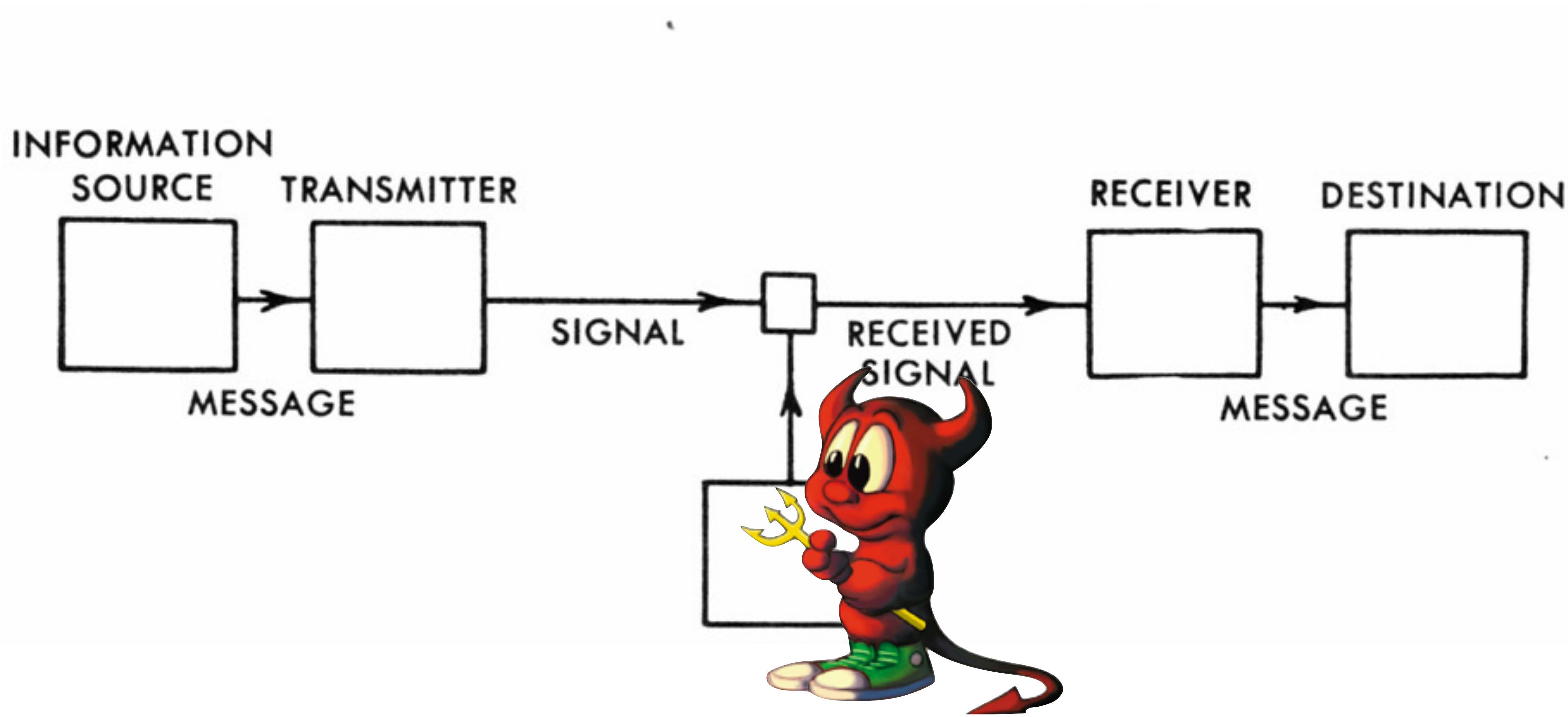
KU LEUVEN

fwo Opening new horizons

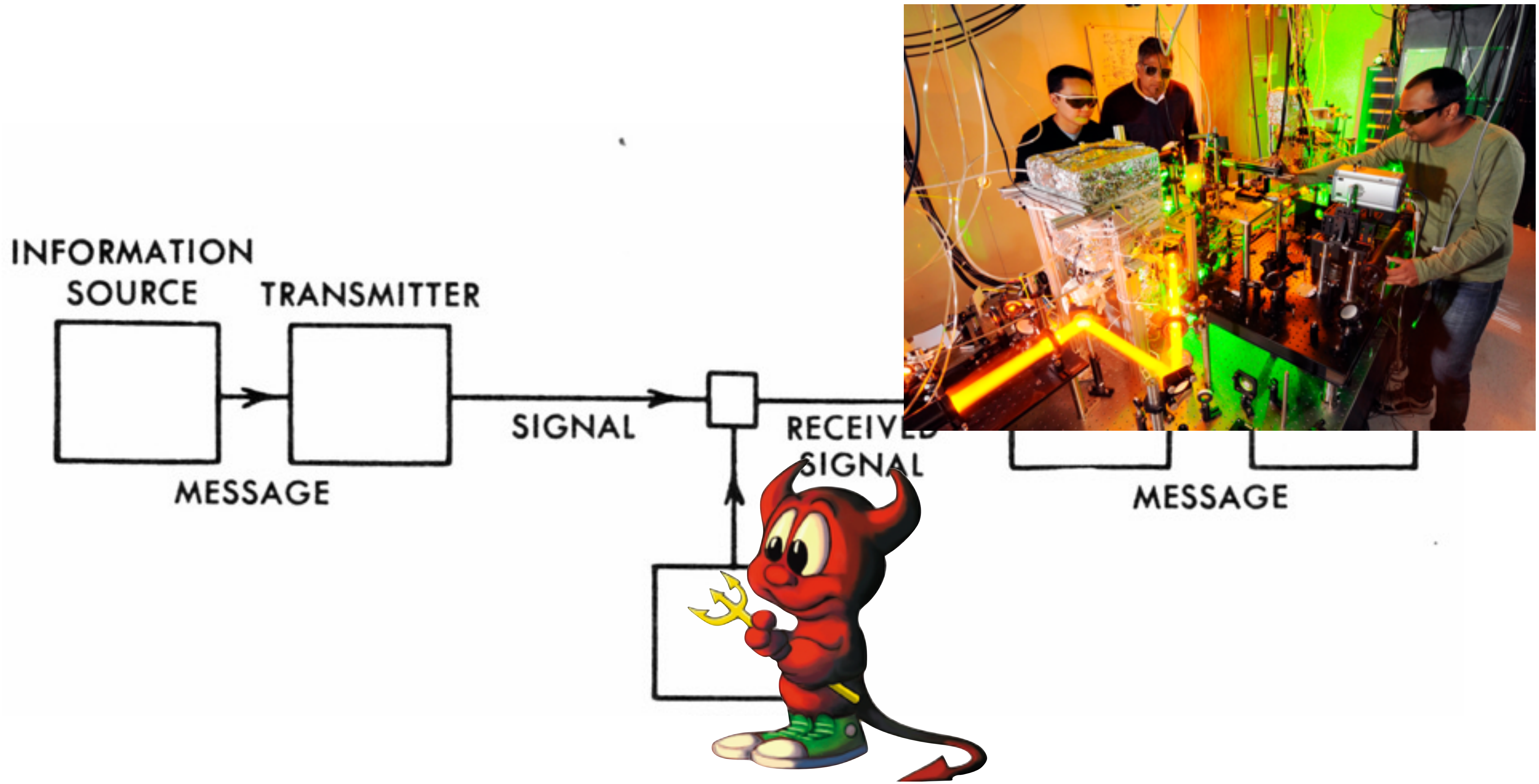
side-channel attacks meets pqcrypto



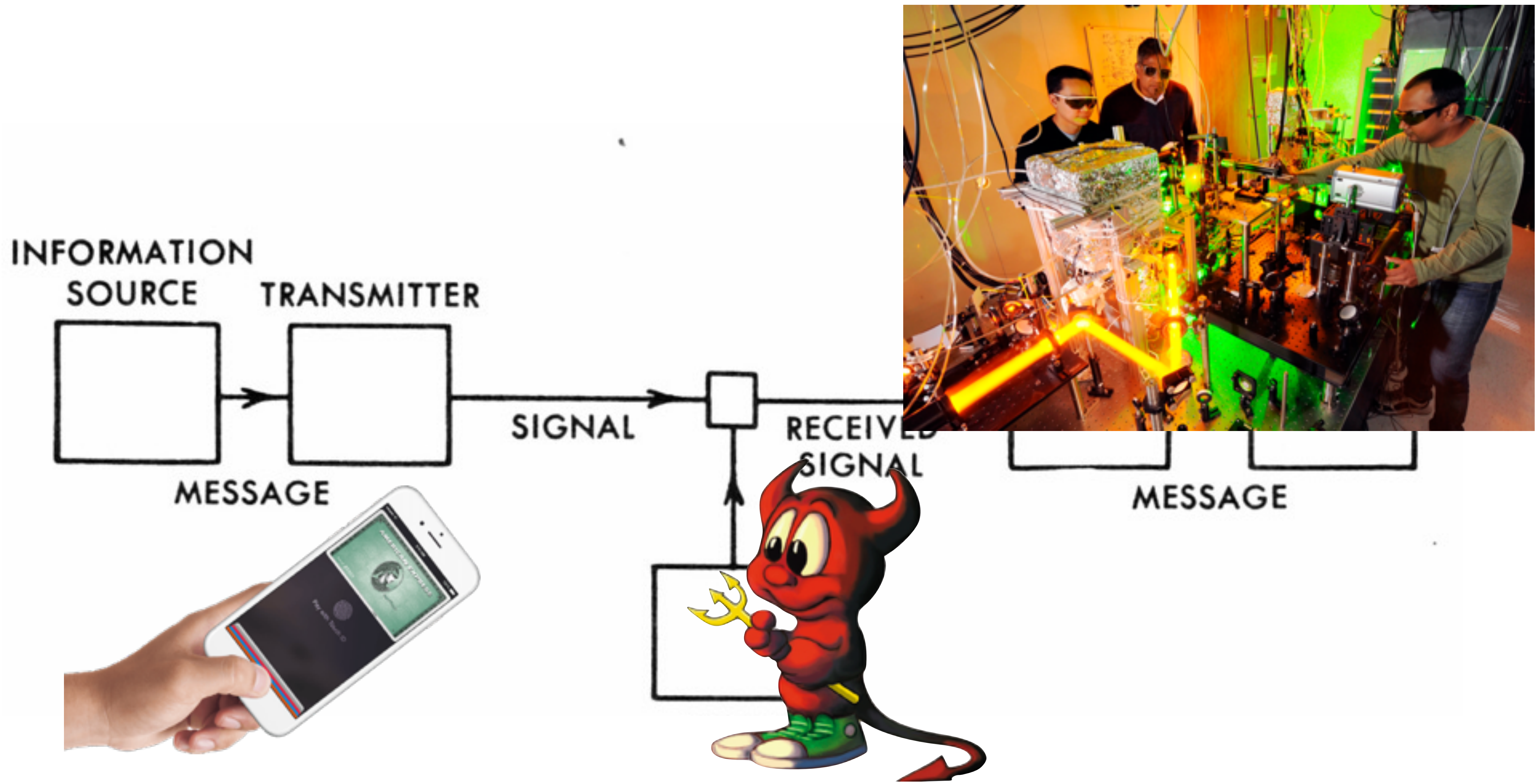
side-channel attacks meets pqcrypto



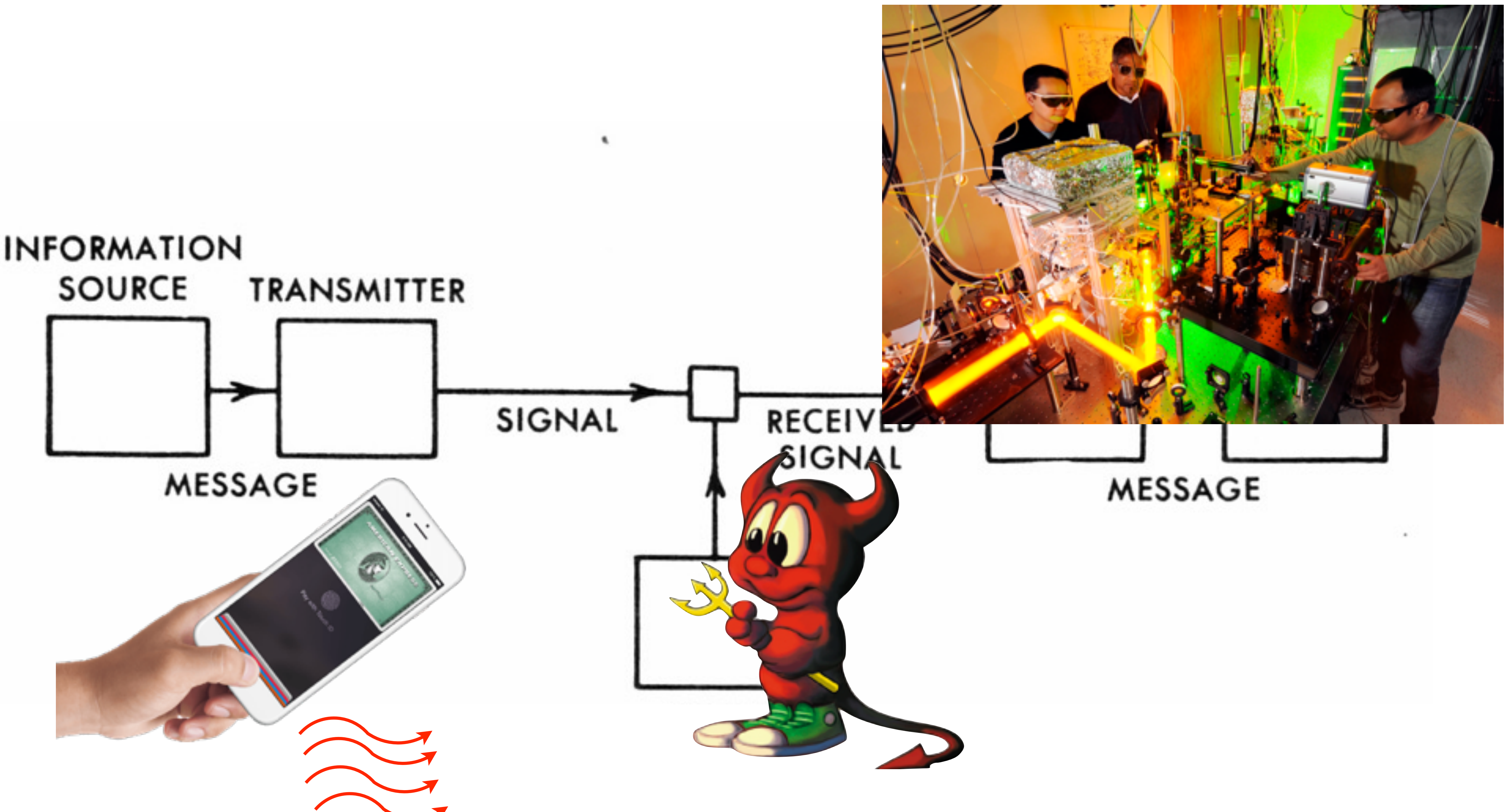
side-channel attacks meets pqcrypto



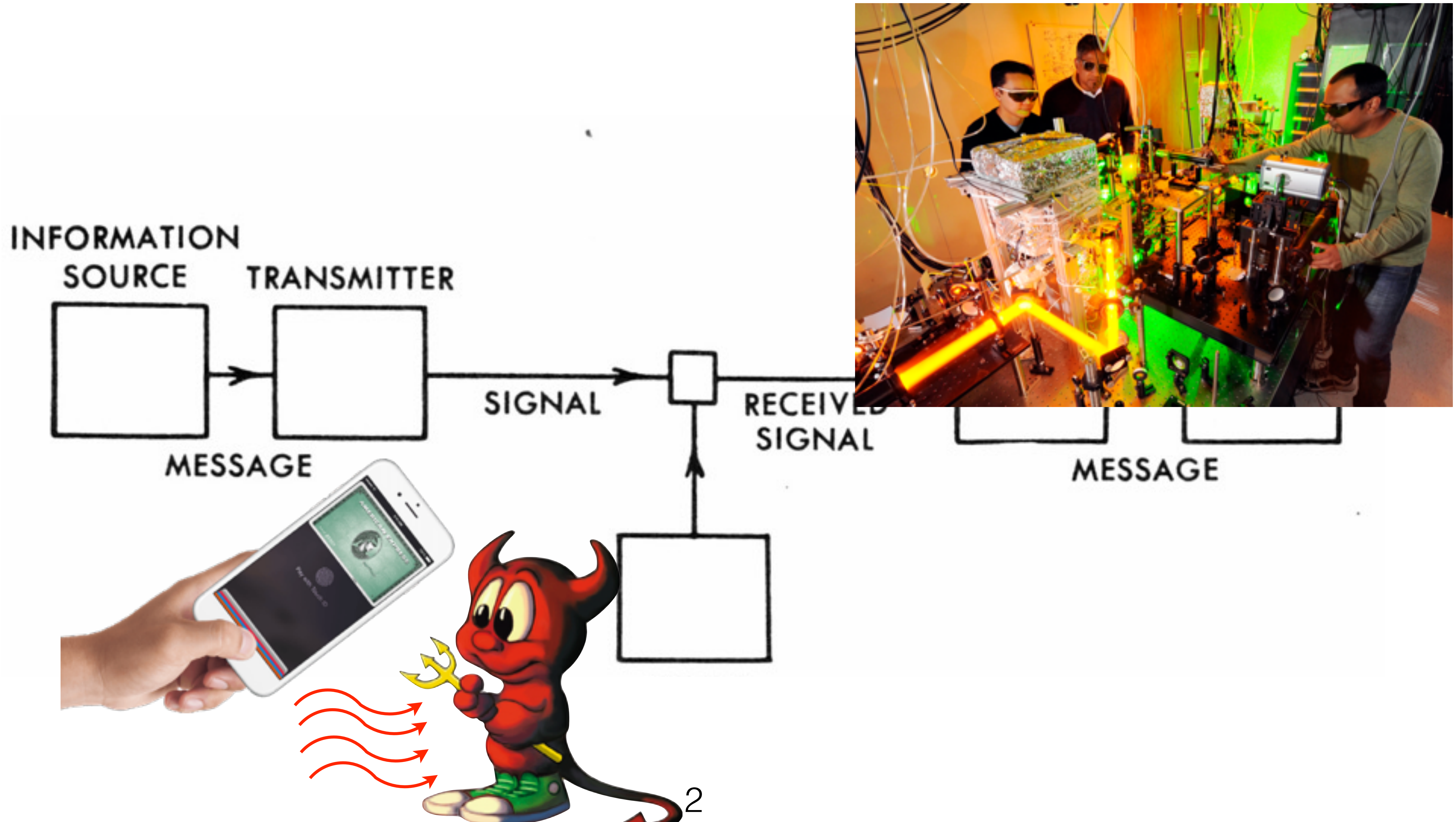
side-channel attacks meets pqcrypto



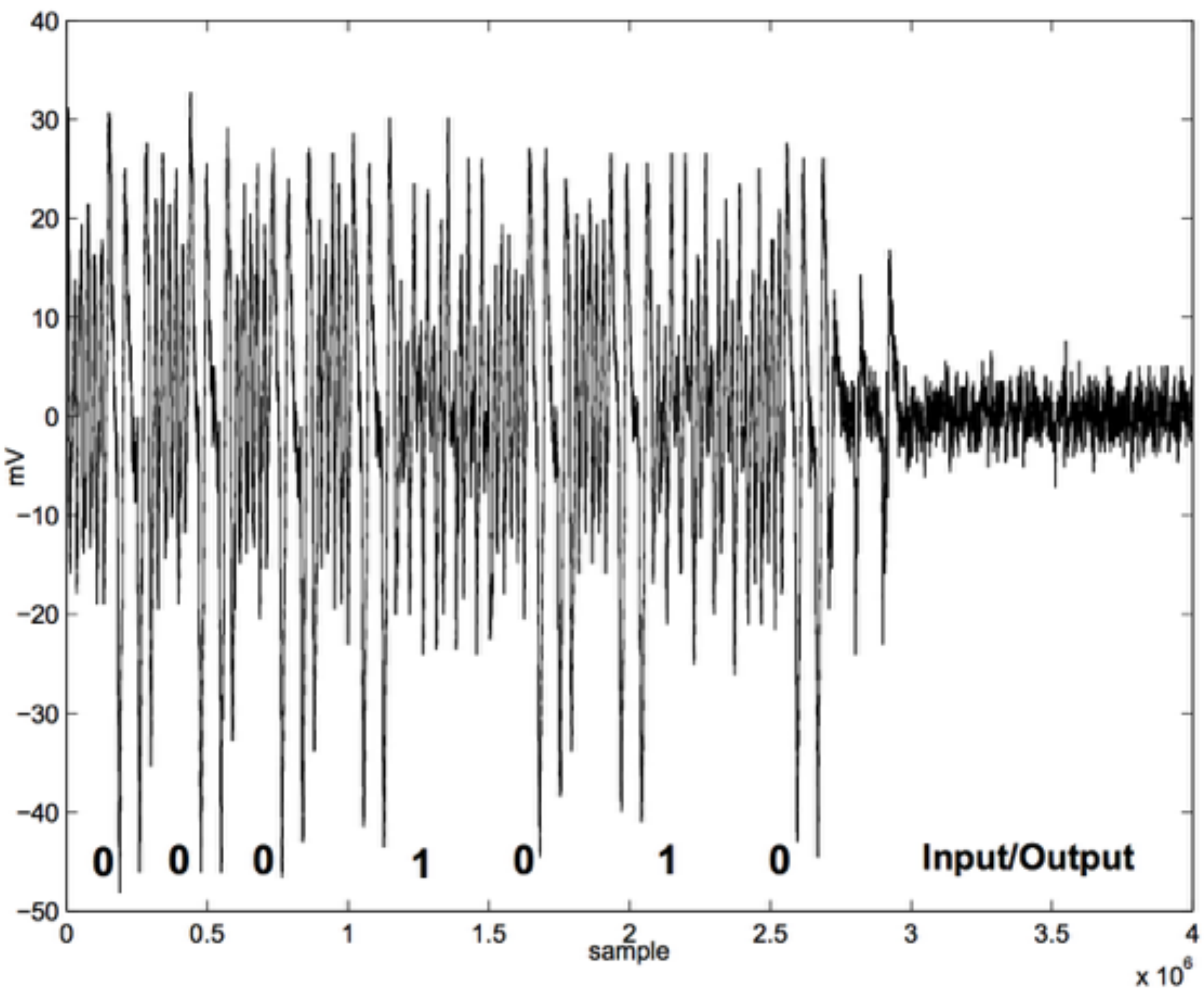
side-channel attacks meets pqcrypto



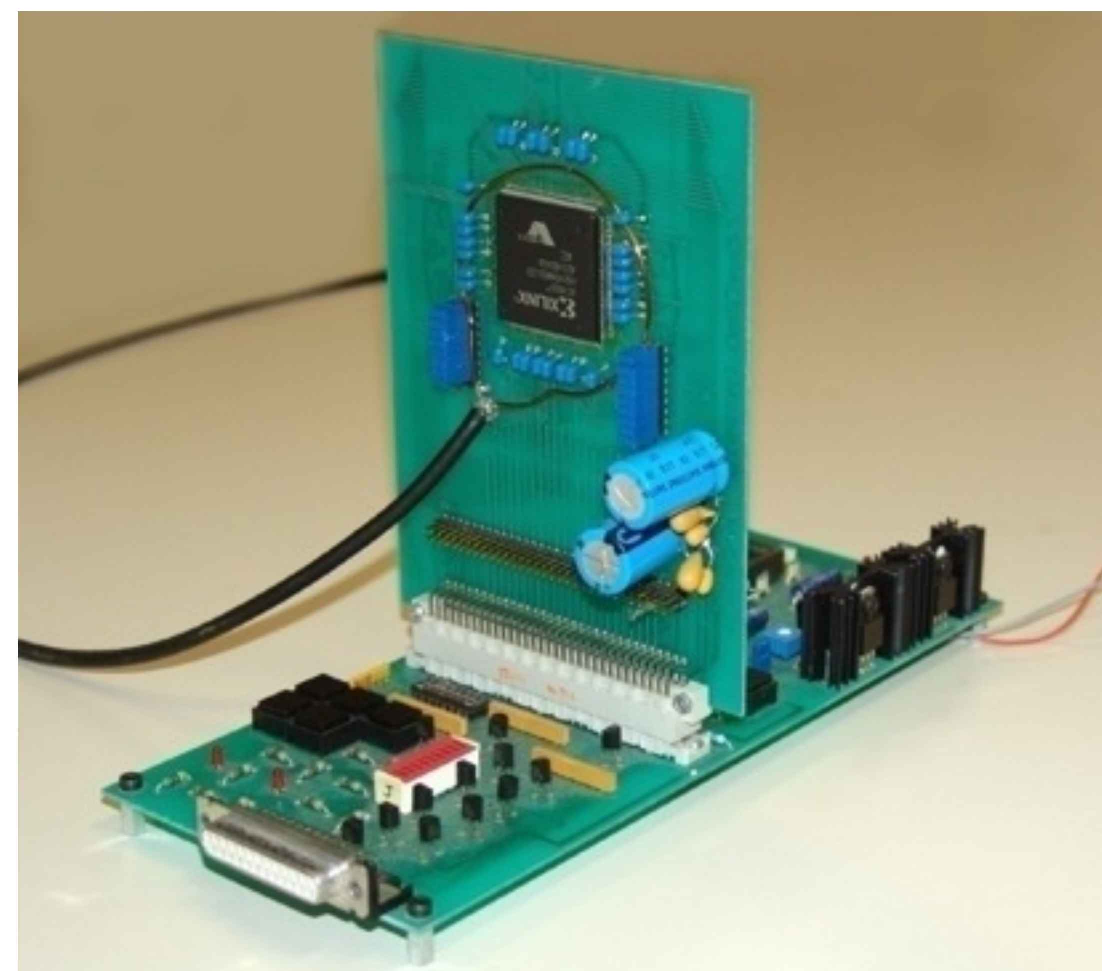
side-channel attacks meets pqcrypto



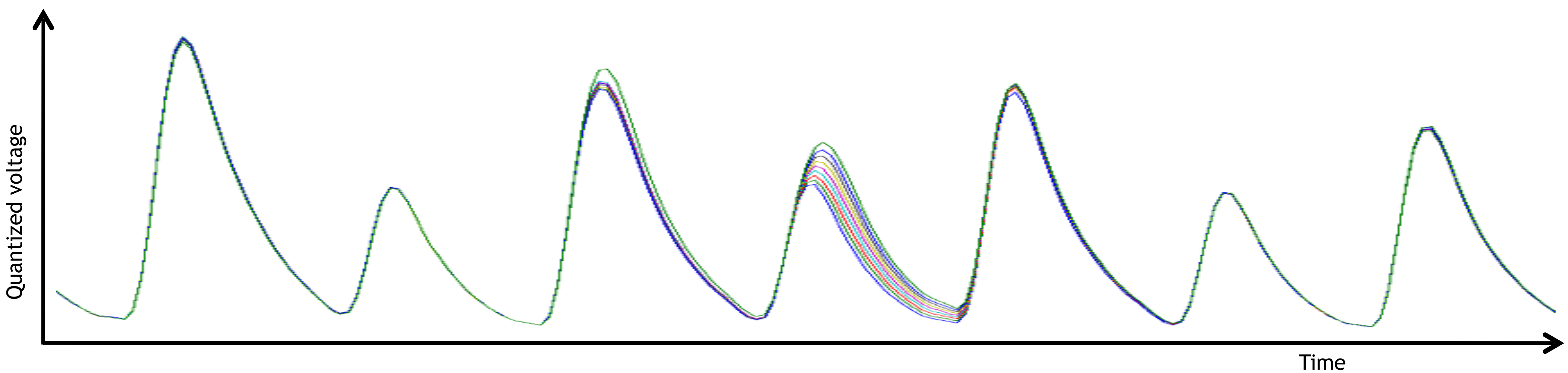
Side-channel leakage example: SPA



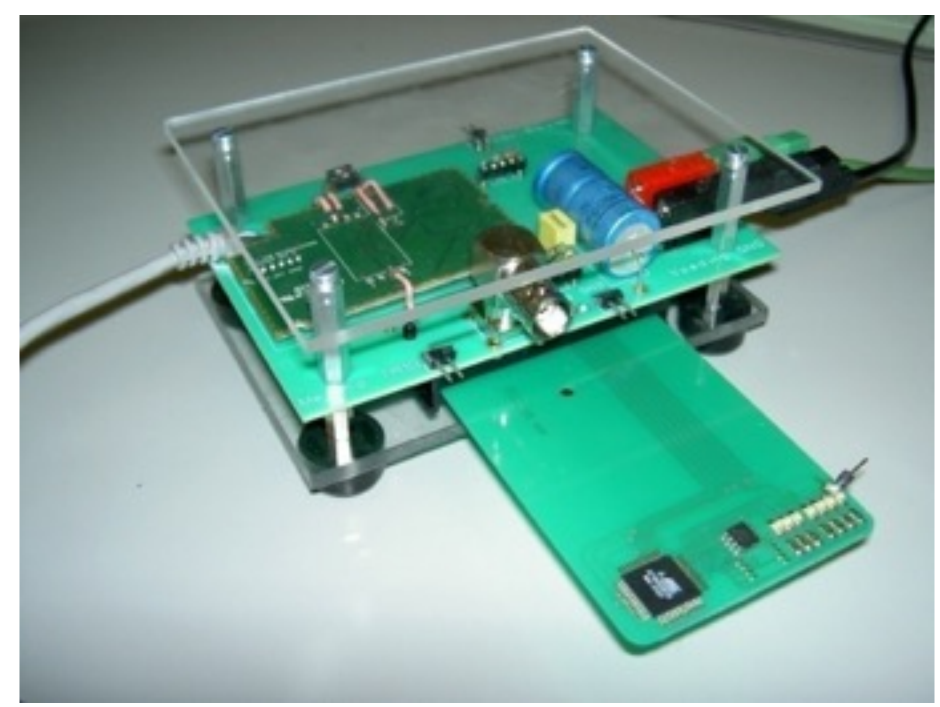
SPA on FPGA ECC (Elke de Mulder)



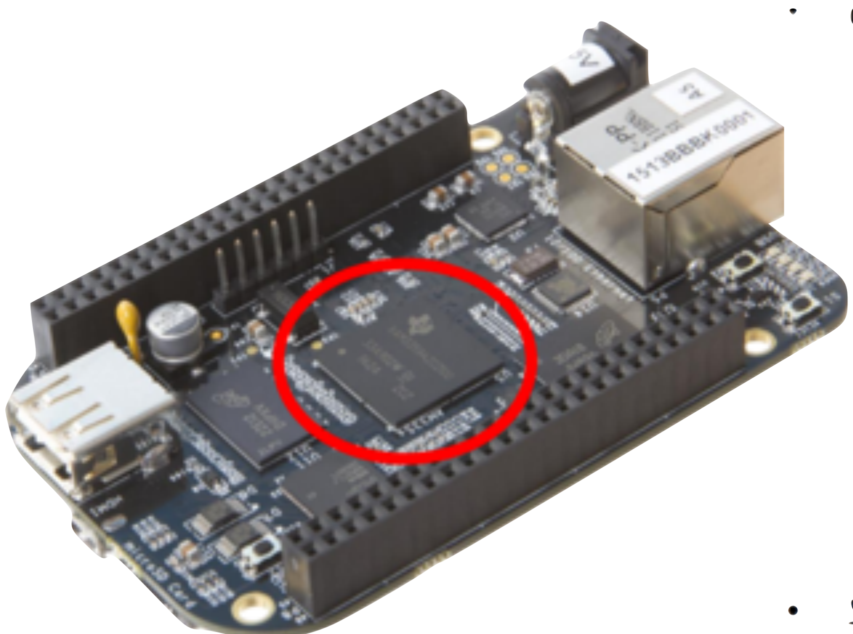
Side-channel leakage example: DPA



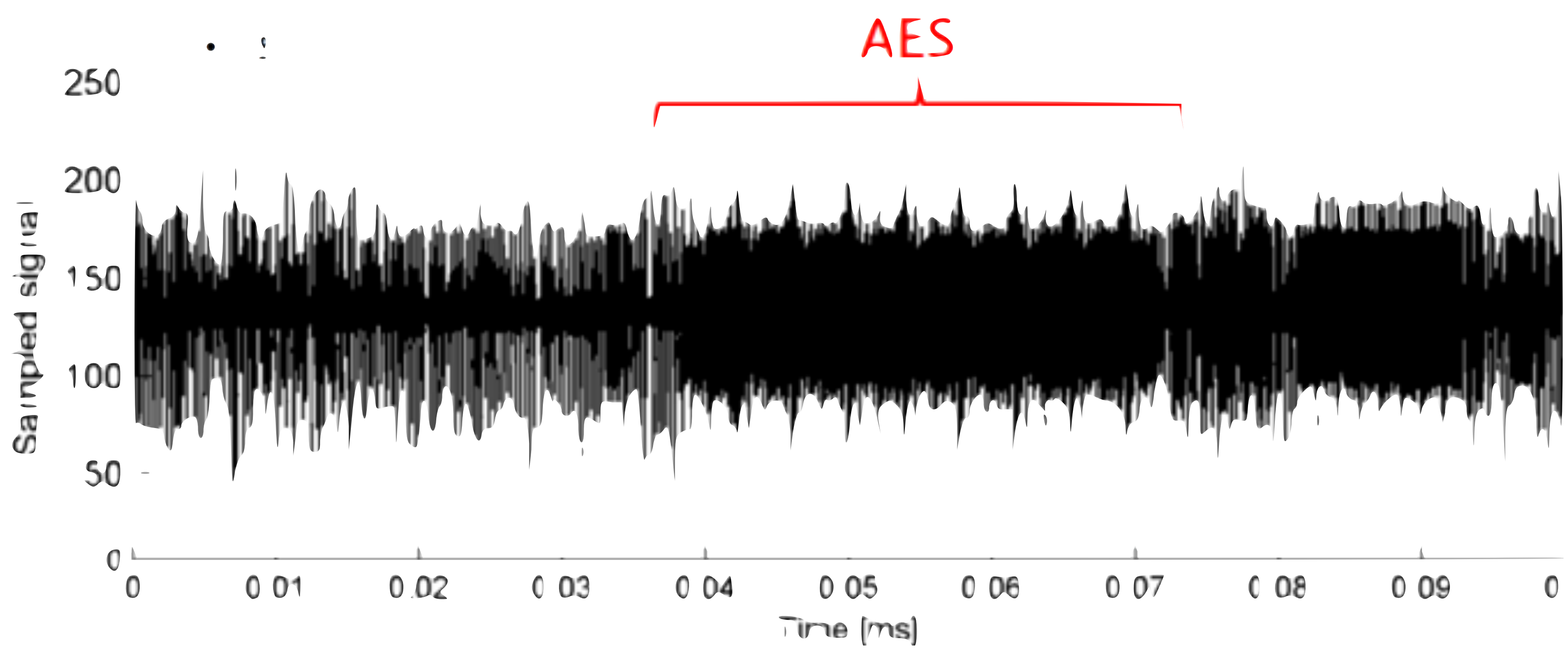
Unprotected 8-bit microcontroller. @3 MHz. Power consumption curves of MOV instruction. Operands with different HW. (J. Balasch)



Side-channel leakage example: DPA



32-bit ARM Cortex-A8 @1 GHz
Linux OS, bitsliced AES



 (Balasch, Gierlichs, Reparaz, Verbauwhede)

masking

- masking = countermeasure against DPA
- idea: secret sharing $b = b_1 + b_2$
- individual shares tell you nothing about the intermediate
  - power consumption tells you nothing about the intermediate
- main difficulty: compute on masked data
  - AES / RSA / ECC / ring-LWE

ring-LWE review

ring-LWE review

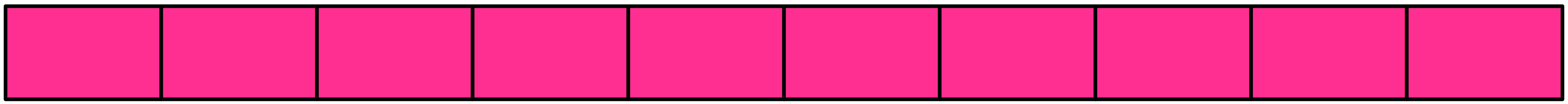
- Works with polynomials in a ring $R_q = \mathbb{Z}_q[\mathbf{x}]/(f(x))$
- Key-gen
  - Sample two polynomials r_1, r_2
    - public key: $p = r_1 - g * r_2$
    - secret key: r_2
- Encryption
  - lift the message to ring element $\bar{m}$
  - compute ciphertext (c_1, c_2)

$$c_1 = g * e_1 + e_2$$
$$c_2 = p * e_1 + e_3 + \bar{m}$$

- Decryption
  - recover message as $m = \mathrm{th}(c_1 * r_2 + c_2)$

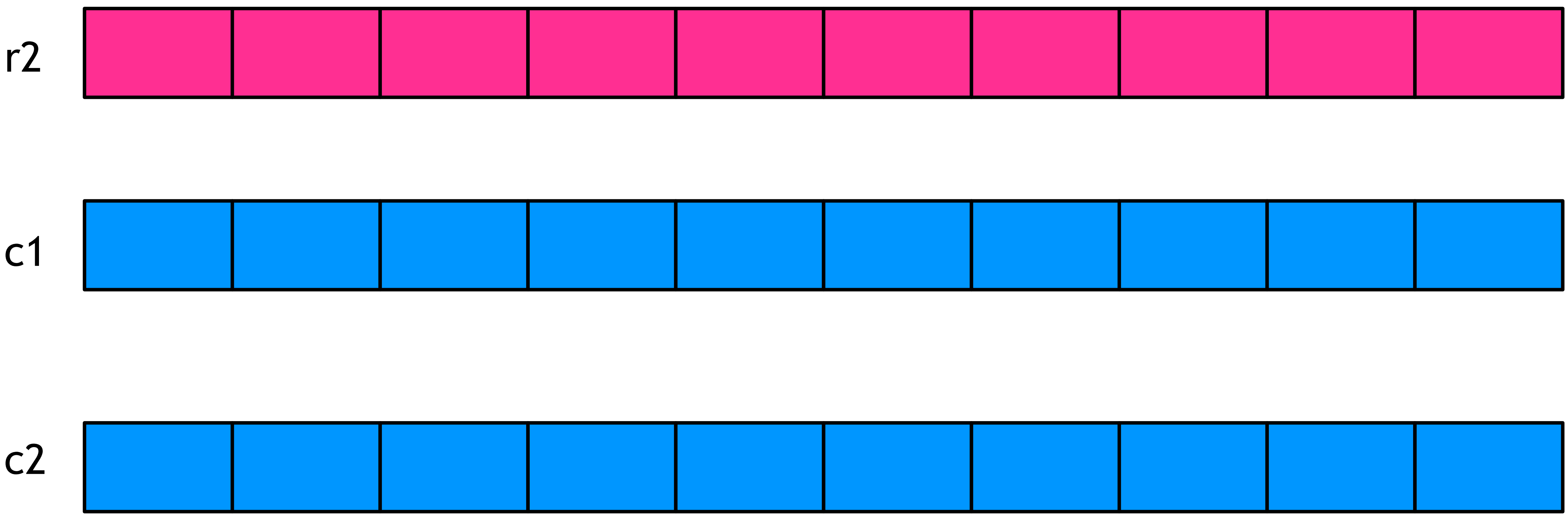
<u>un</u>protected ring-LWE decryption

r2



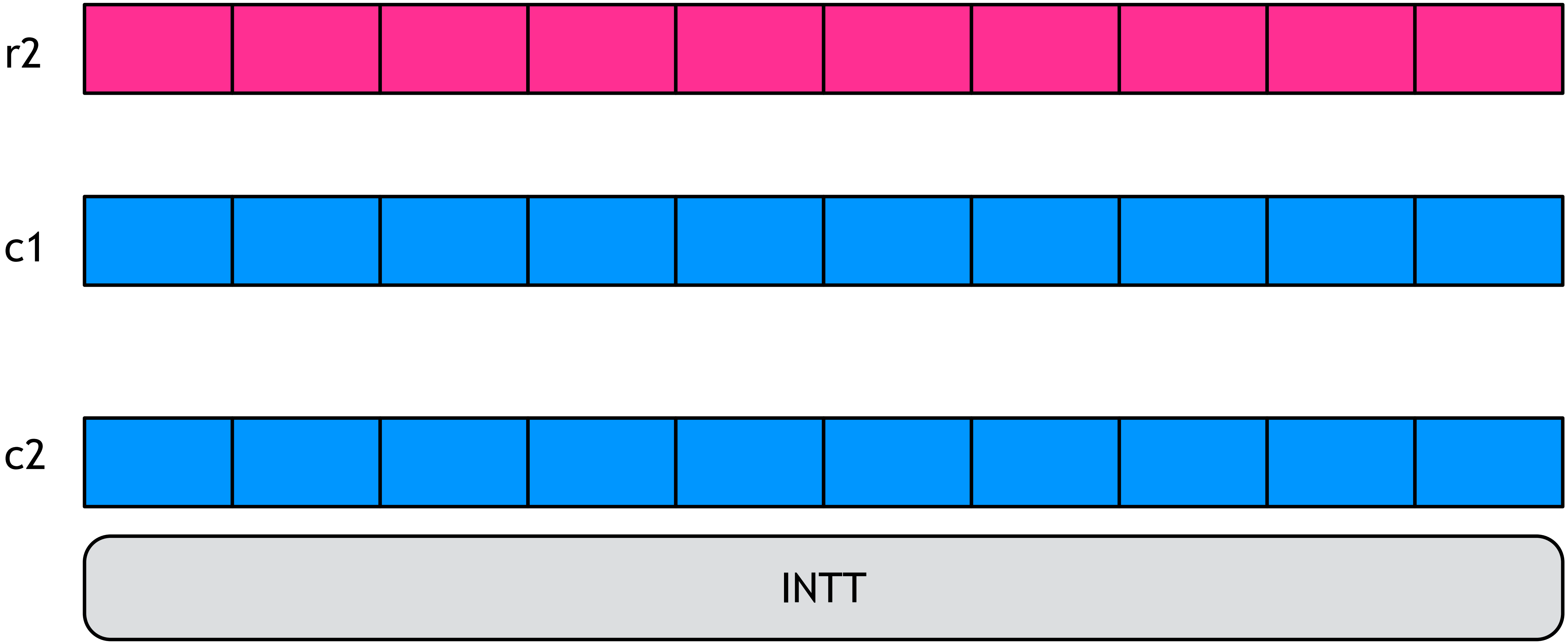
$$m = \mathrm{th}[\mathrm{INTT}(c_1 * r_2 + c_2)]$$

<u>un</u>protected ring-LWE decryption



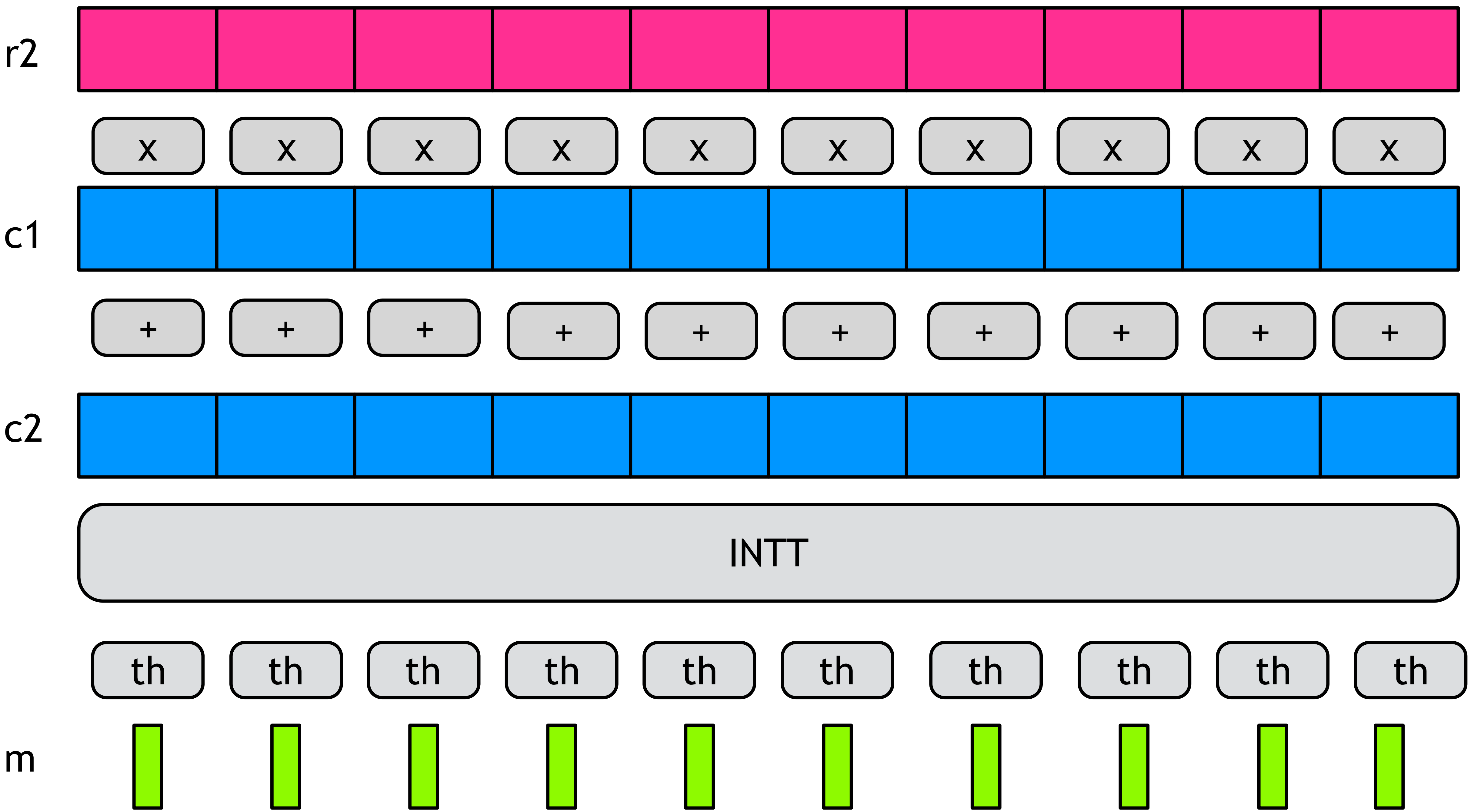
$$m = \mathrm{th}[\mathrm{INTT}(c_1 * r_2 + c_2)]$$

<u>**un**</u>protected ring-LWE decryption



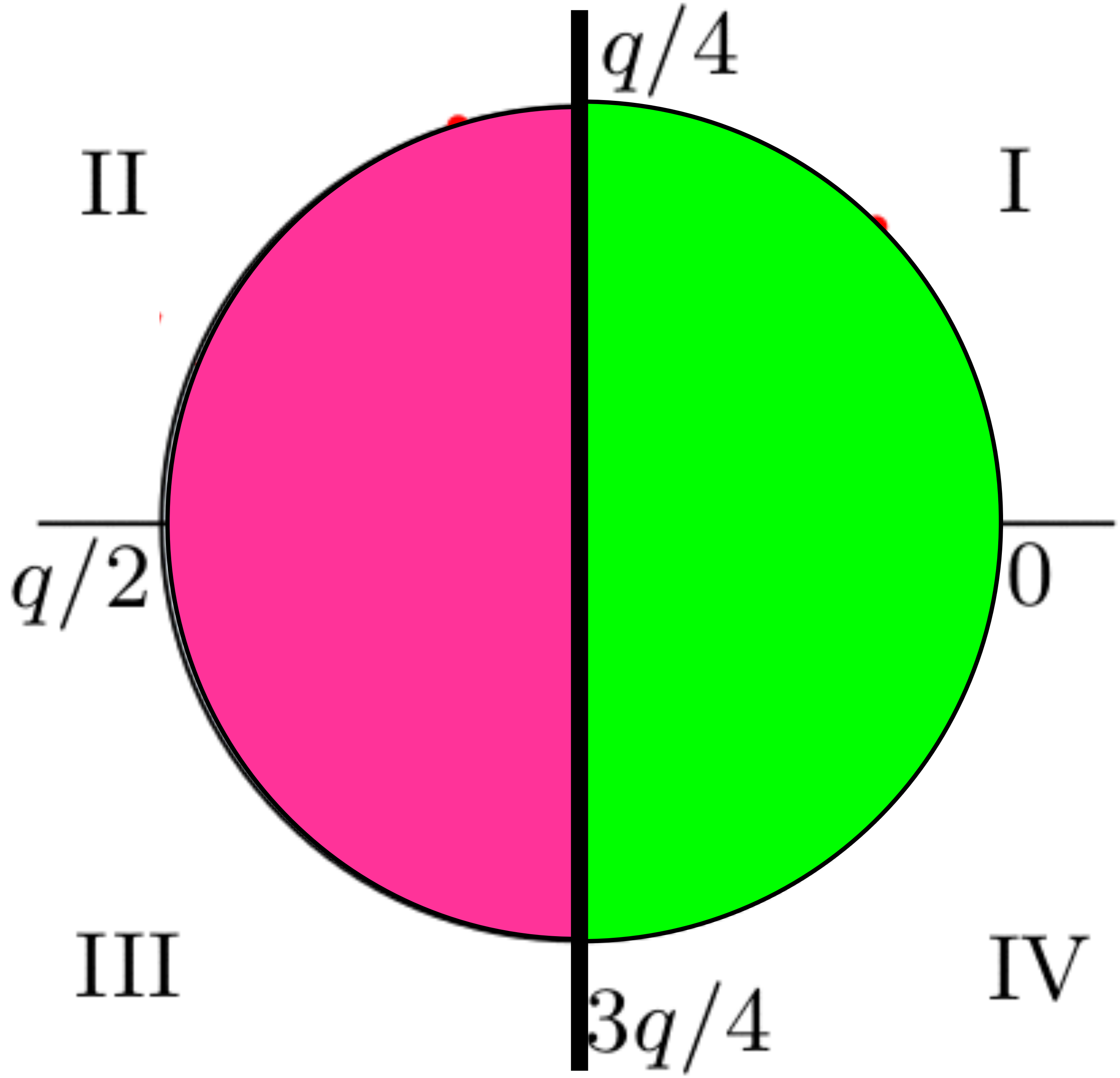
$$m = \mathrm{th}[\mathrm{INTT}(c_1 * r_2 + c_2)]$$

unprotected ring-LWE decryption



$$m = \mathrm{th}[\mathrm{INTT}(c_1 * r_2 + c_2)]$$

th operation



masking ring-LWE

Two approaches:

1. Split the key
   "A masked ring-LWE implementation", CHES 2015
   "Masking ring-LWE", JCEN

2. Split the input
   "Additively Homomorphic ring-LWE Masking" PQCrypto 2016

CHES 2015: split the key

- Core idea: split the secret: r=r’+r”

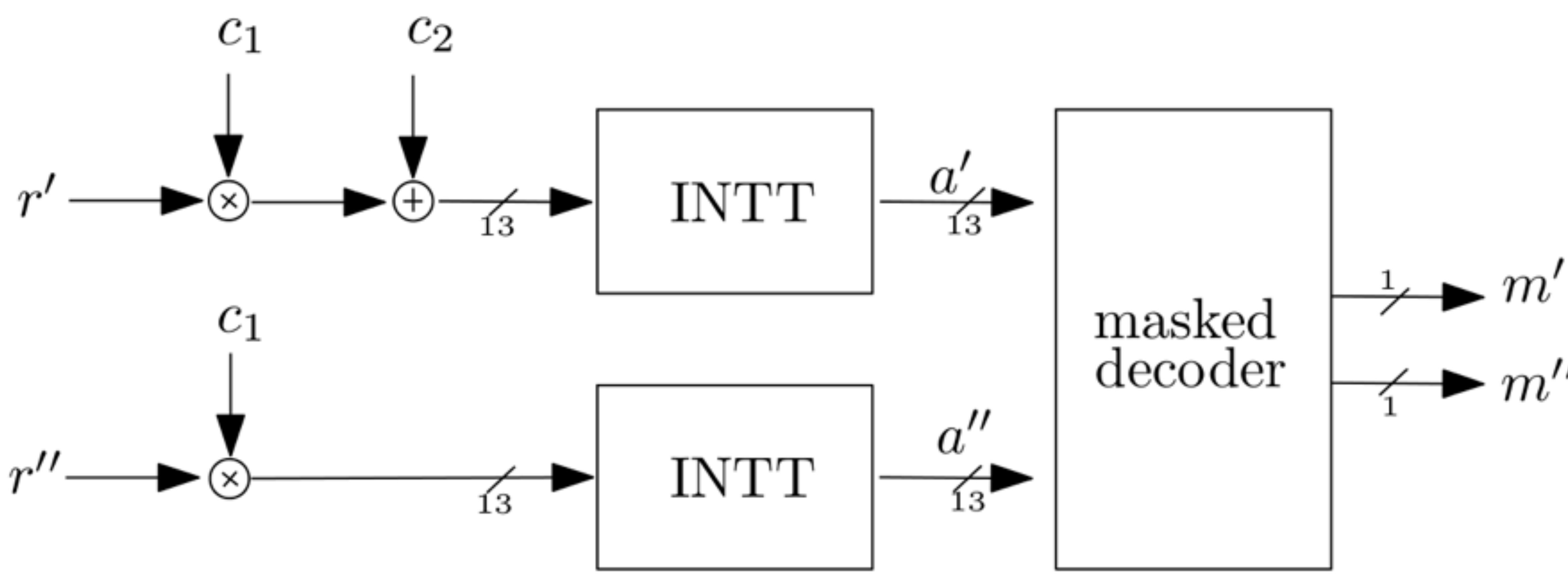
$$\mathrm{INTT}(r \cdot c_2 + c_1) = \mathrm{INTT}(r' \cdot c_2 + c_1) + \mathrm{INTT}(r'' \cdot c_2).$$

$m=th[\mathrm{INTT}(c_1 * r_2 + c_2)]$

CHES 2015: split the key

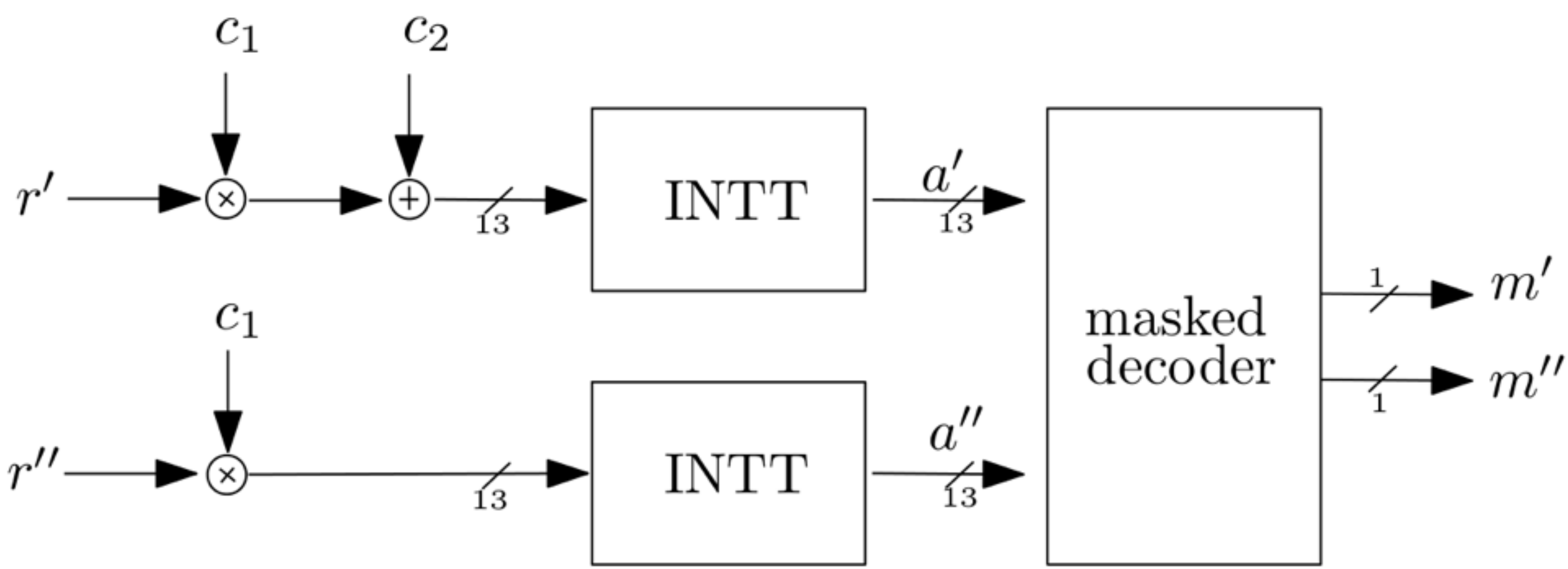
- Core idea: split the secret: r=r’+r’’

$$\mathrm{INTT}(r \cdot c_2 + c_1) \;=\; \mathrm{INTT}(r' \cdot c_2 + c_1) + \mathrm{INTT}(r'' \cdot c_2).$$

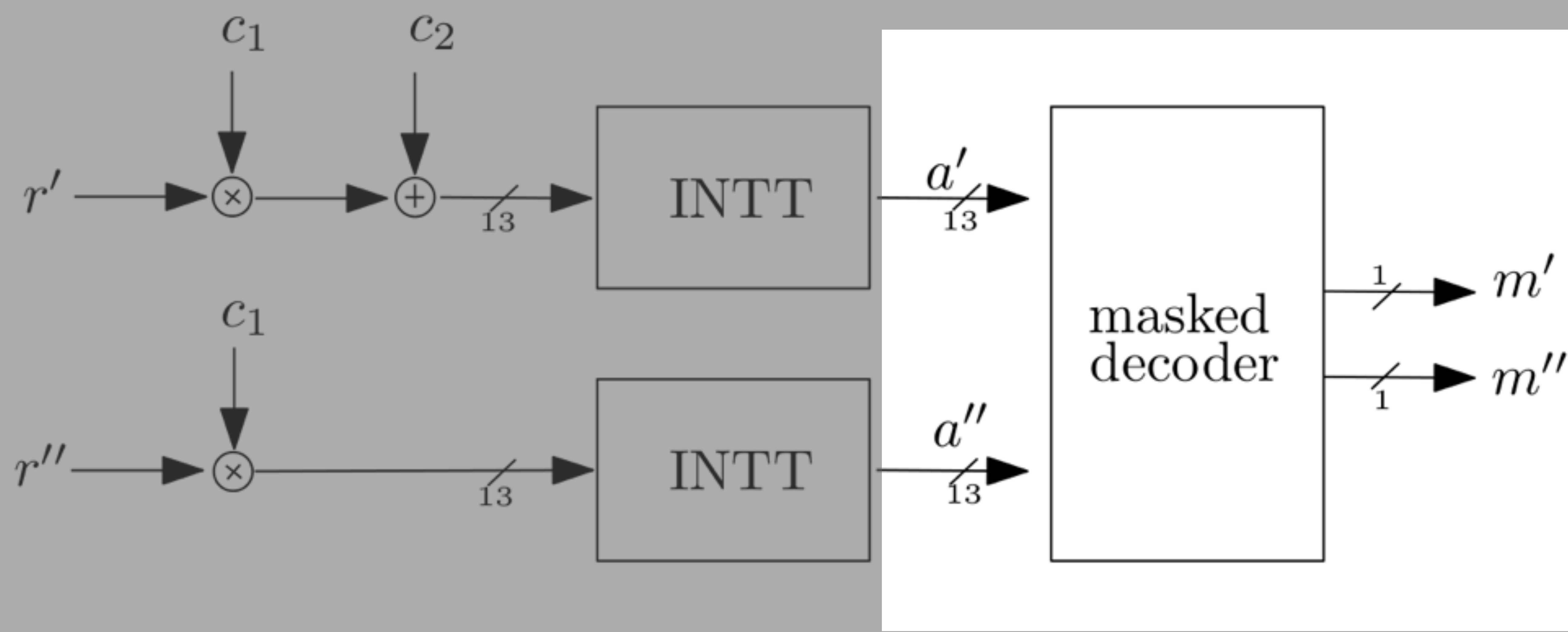


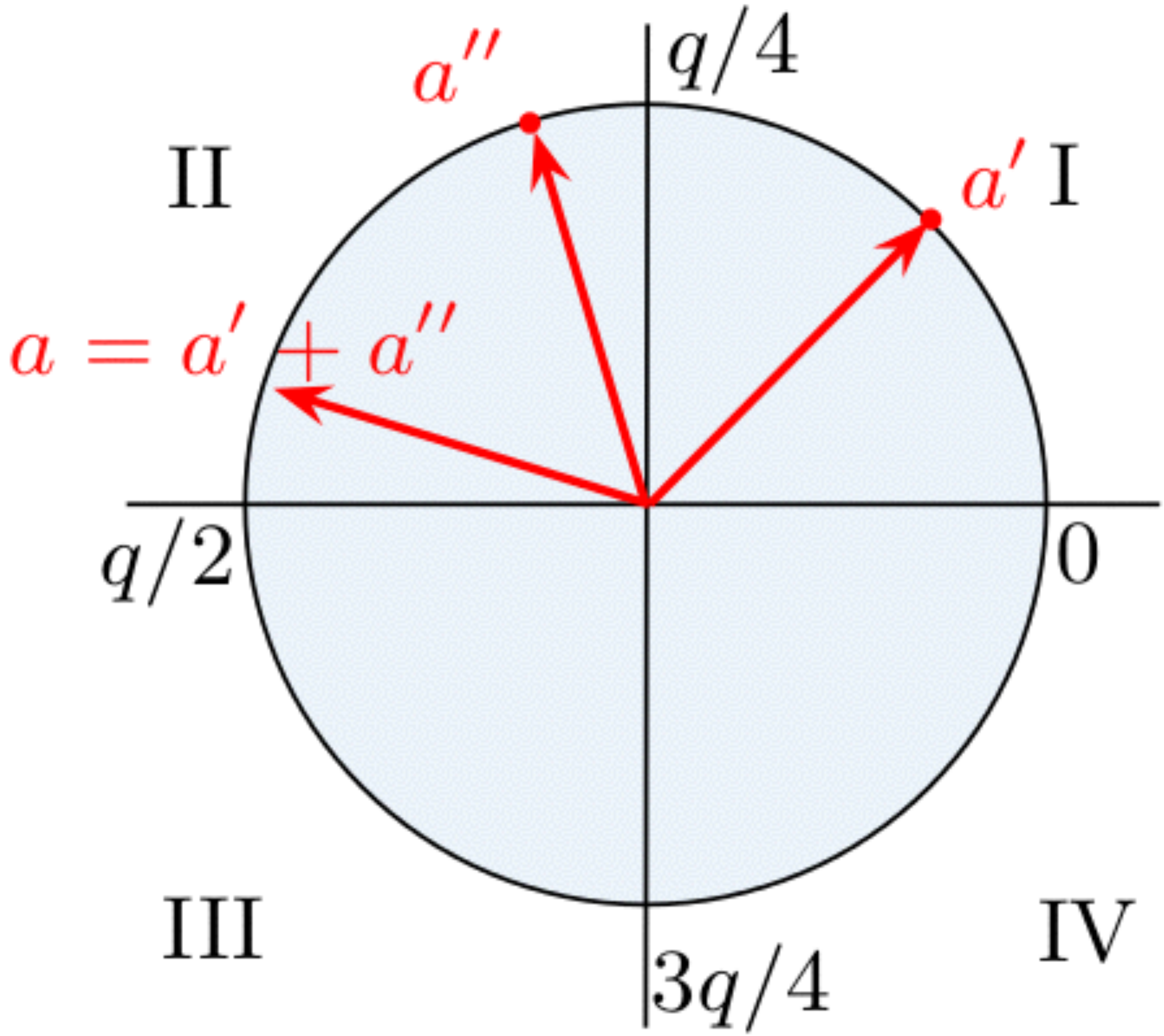
$$m=th[\mathrm{INTT}(c_1*r_2+ c_2)]$$

on the masked decoder

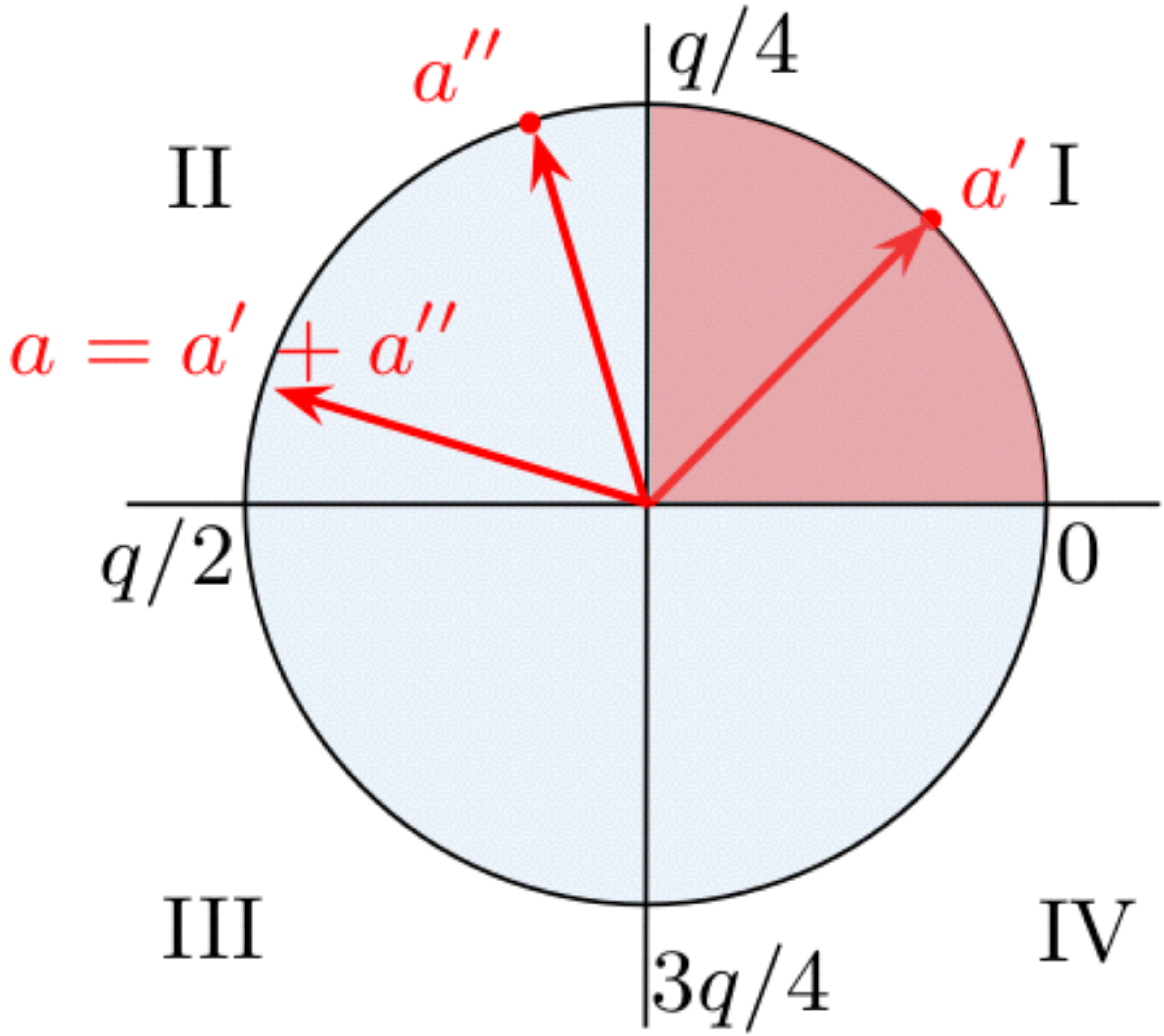


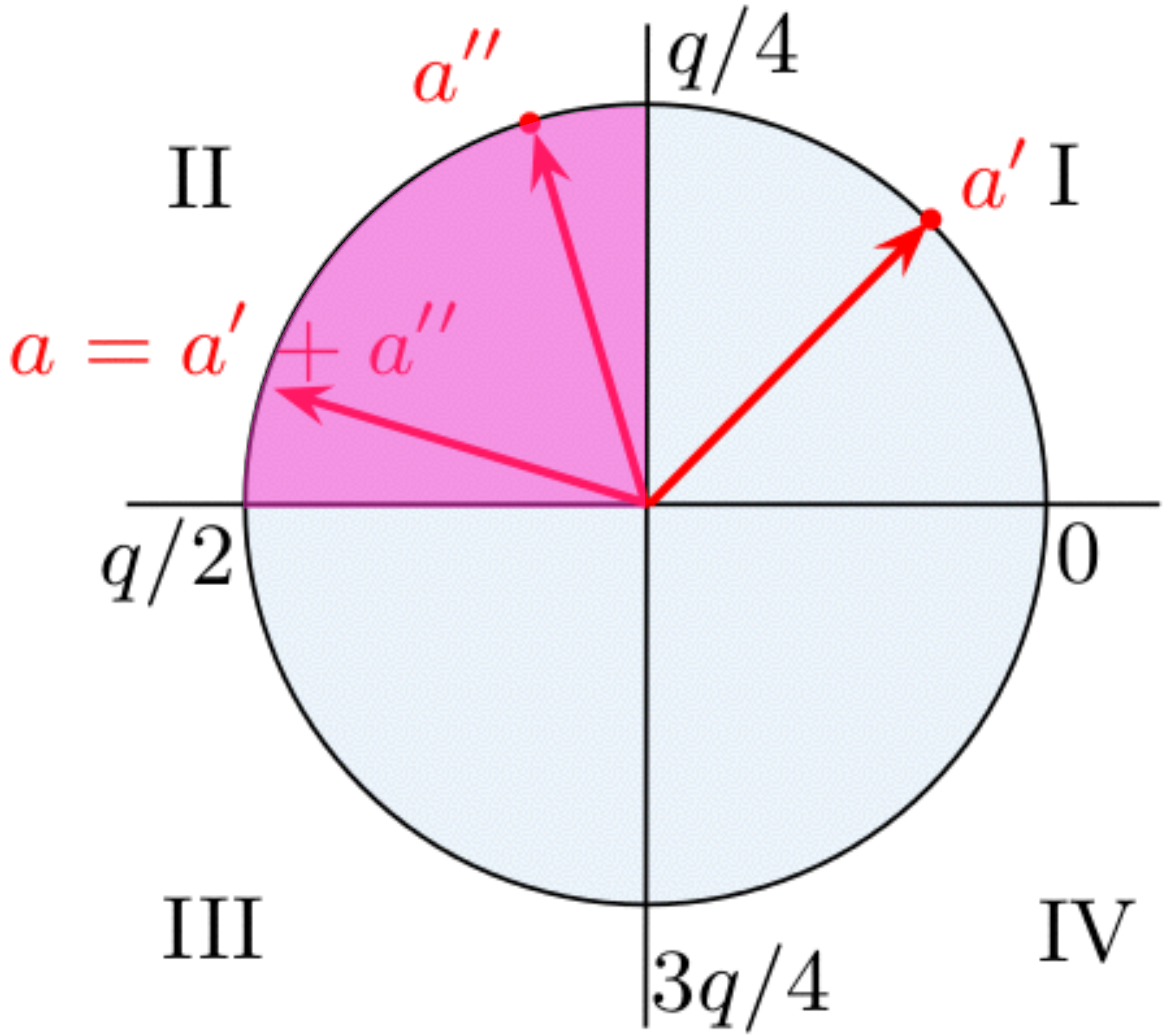
on the masked decoder

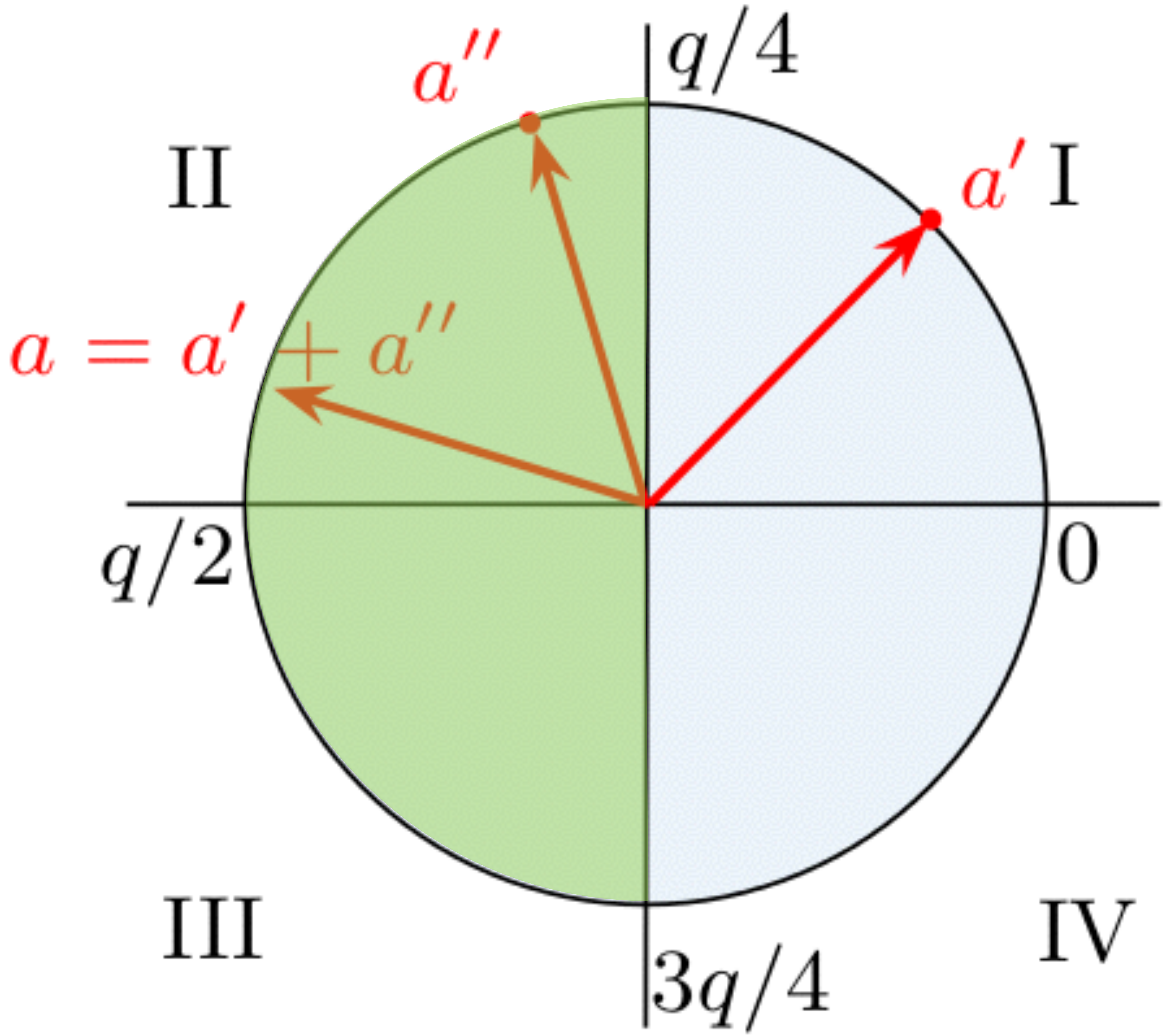




$q/4$
a''
II
a'
I
$a = a' + a''$
$q/2$
0
III
IV
$3q/4$


$q/4$
a''
II
a'
I
$a = a' + a''$
$q/2$
0
III
IV
$3q/4$


$q/4$
a''
II
I
a'
$a = a' + a''$
$q/2$
0
III
IV
$3q/4$



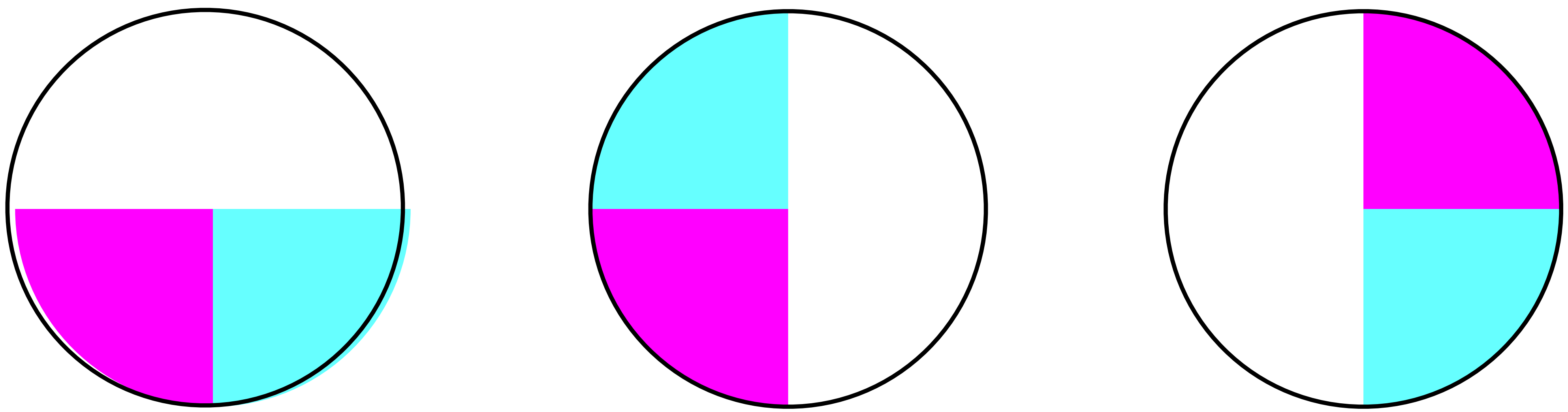
q/4
a''
II
a' I
a = a' + a''
q/2
0
III
IV
3q/4

what happened?

- could decode th(a) from quad(a’) and quad(a’’)
    - quad() return only 2 bits, so it will be easy to perform masked computation.
- Idea: decode th(a) only from quad(a’) and quad(a’’)
    - large compression

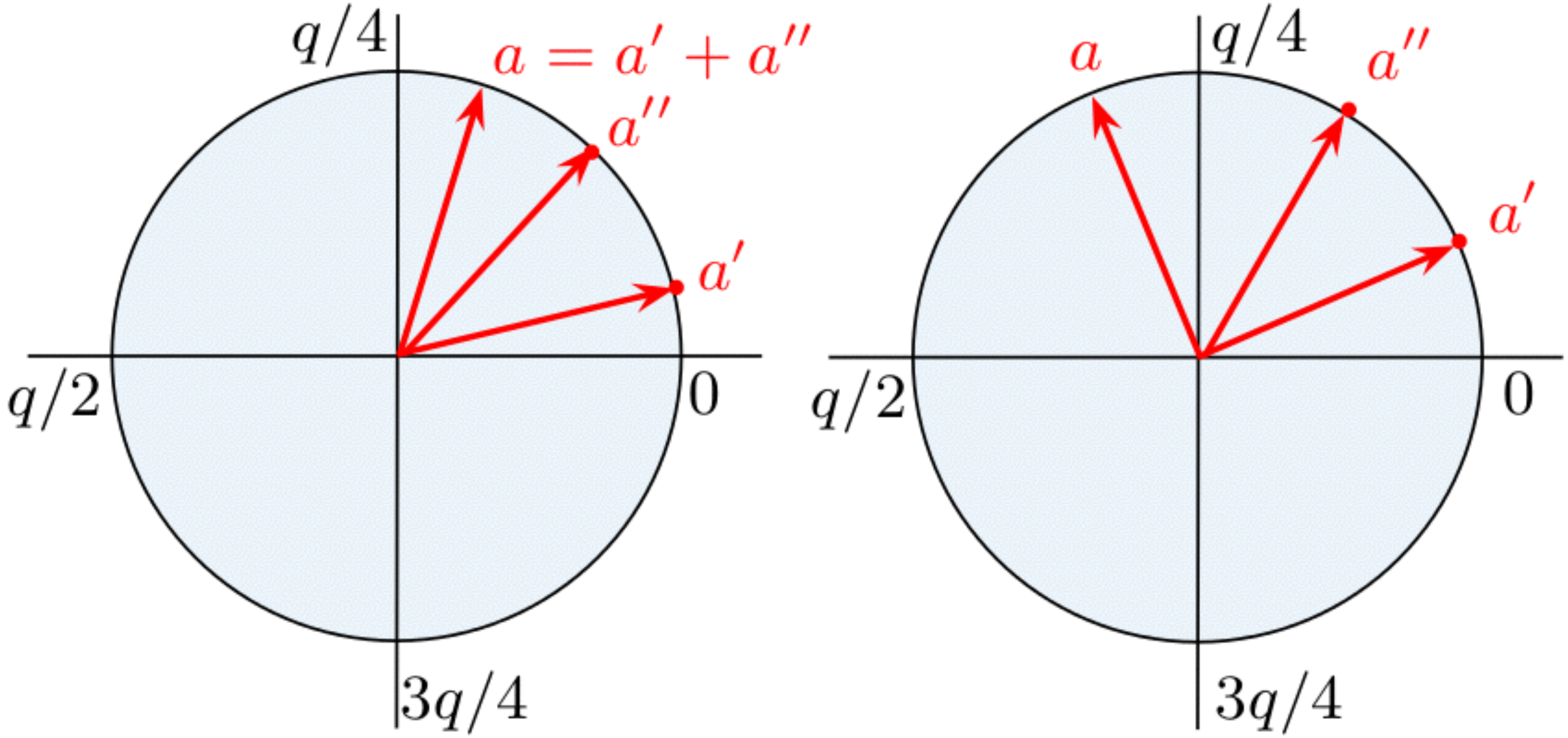
decoding rules

- There are 7 other more cases ("rules")



- There are 8 cases that don't allow inferring th(a)!

Cases where it fails



Solution: refresh the sharing and try again.

$$a' := a' + \Delta$$

$$a'' := a'' - \Delta \quad \text{(compute nice } \Delta \text{)}$$

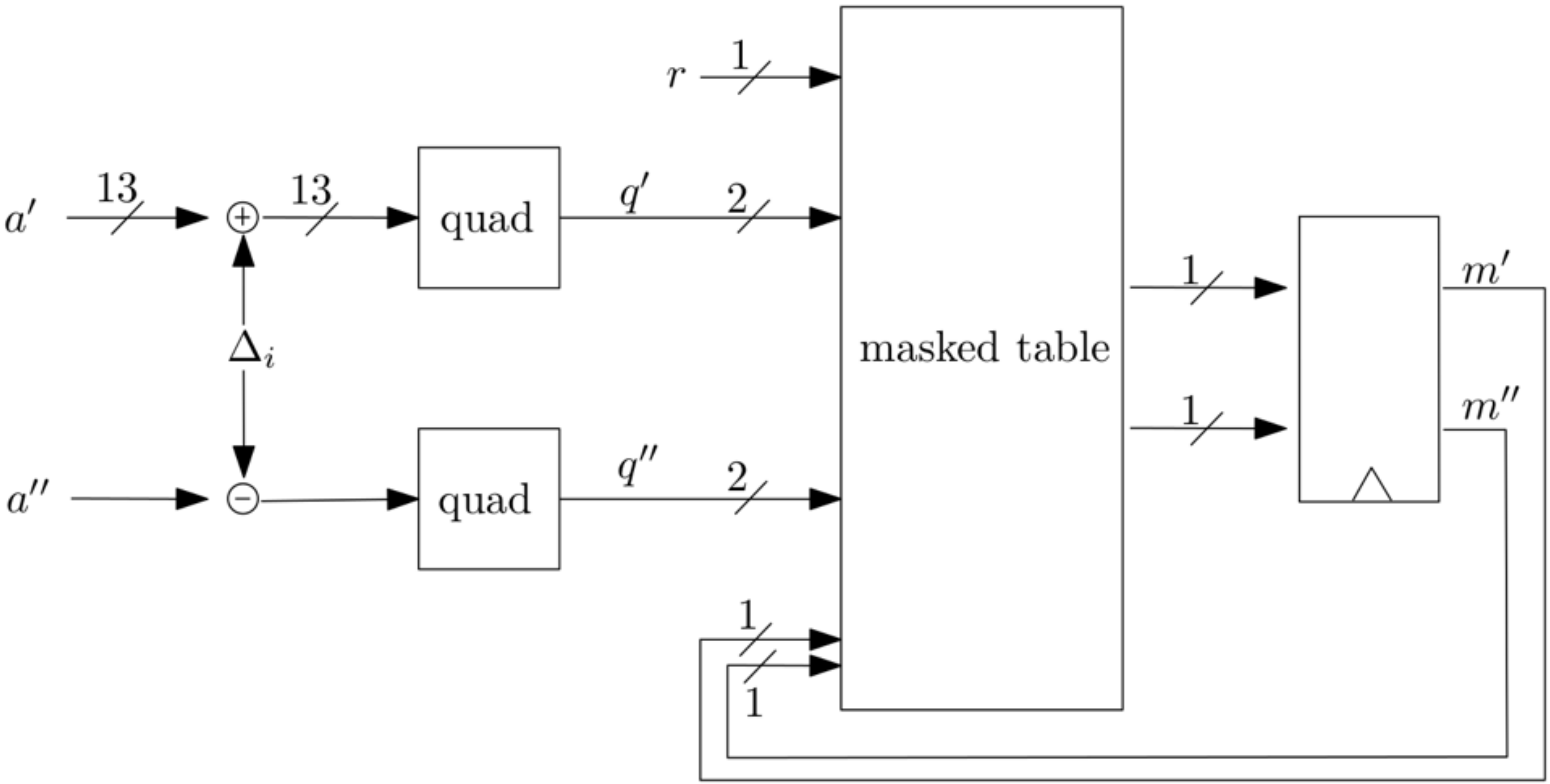


Fig. 3: The masked decoder.

implementation costs

unprotected (CHES2014*)

- 1713 LUTs / 830 FFs / 1 DSP
- Fmax = 120 MHz
- 2.8 k cycles (23.5 us)

protected (this work)

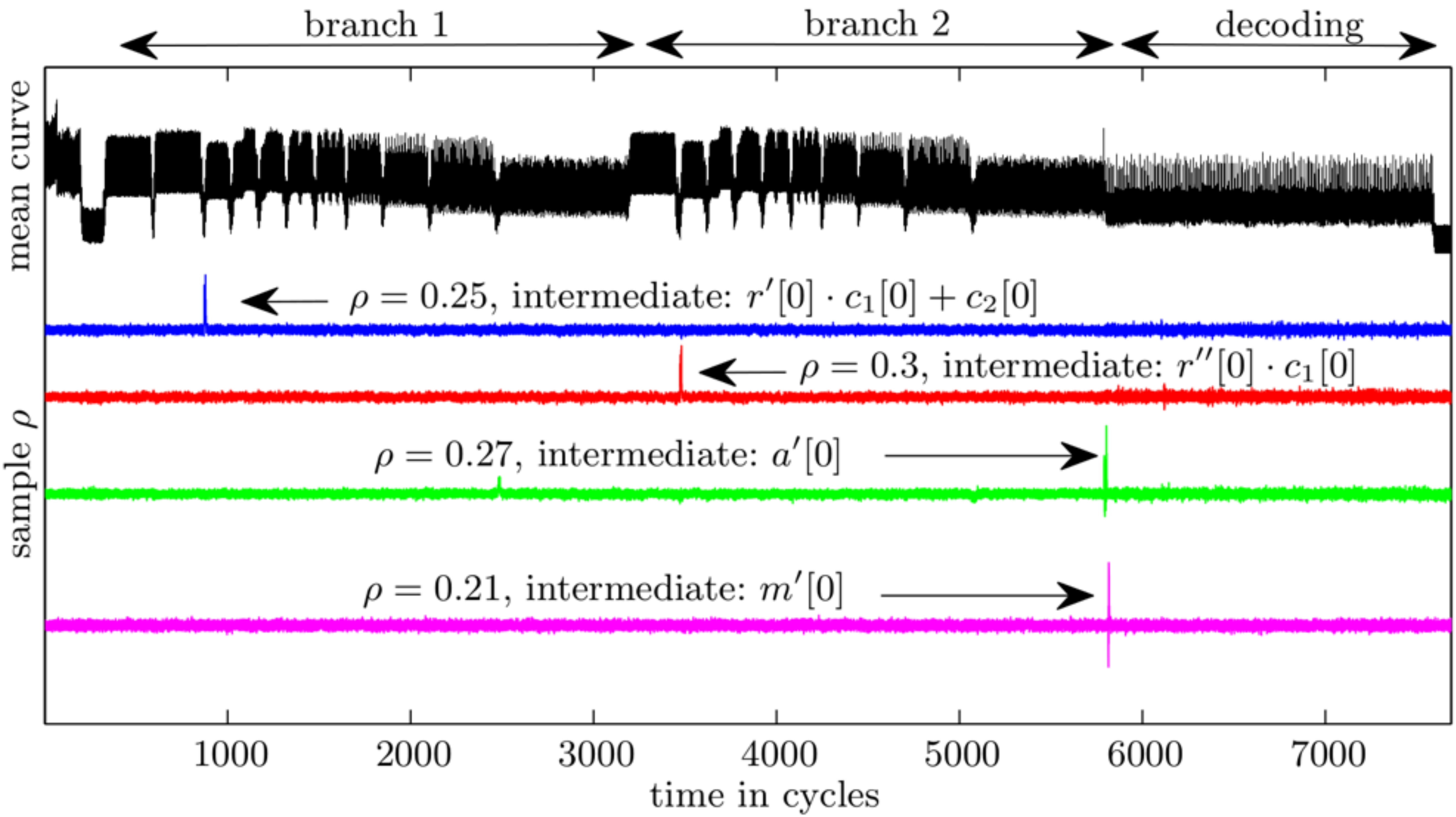
- 2014 LUTs / 959 FFs / 1 DSP
- 100 MHz
- 7.5 k cycles (75.2 us)

Parameter set: (n,q,s)=(256,7681,11.32)
Xilinx Virtex-II xc2vp7 FPGA

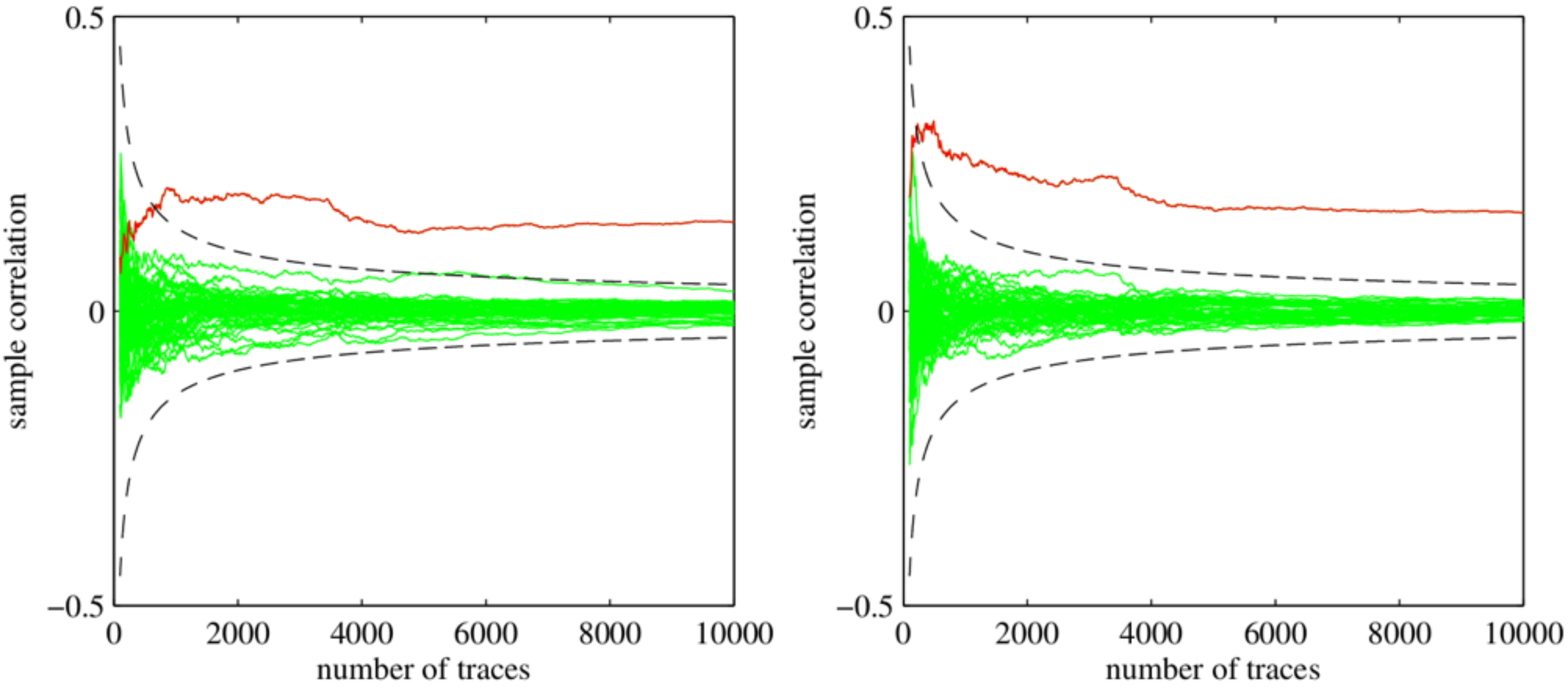
ECC: Rebeiro et.al. (CHES2012): 289 kcycles * LUT
This work: 151 k cycles*LUTs

* Synthetized on Virtex-II

evaluation

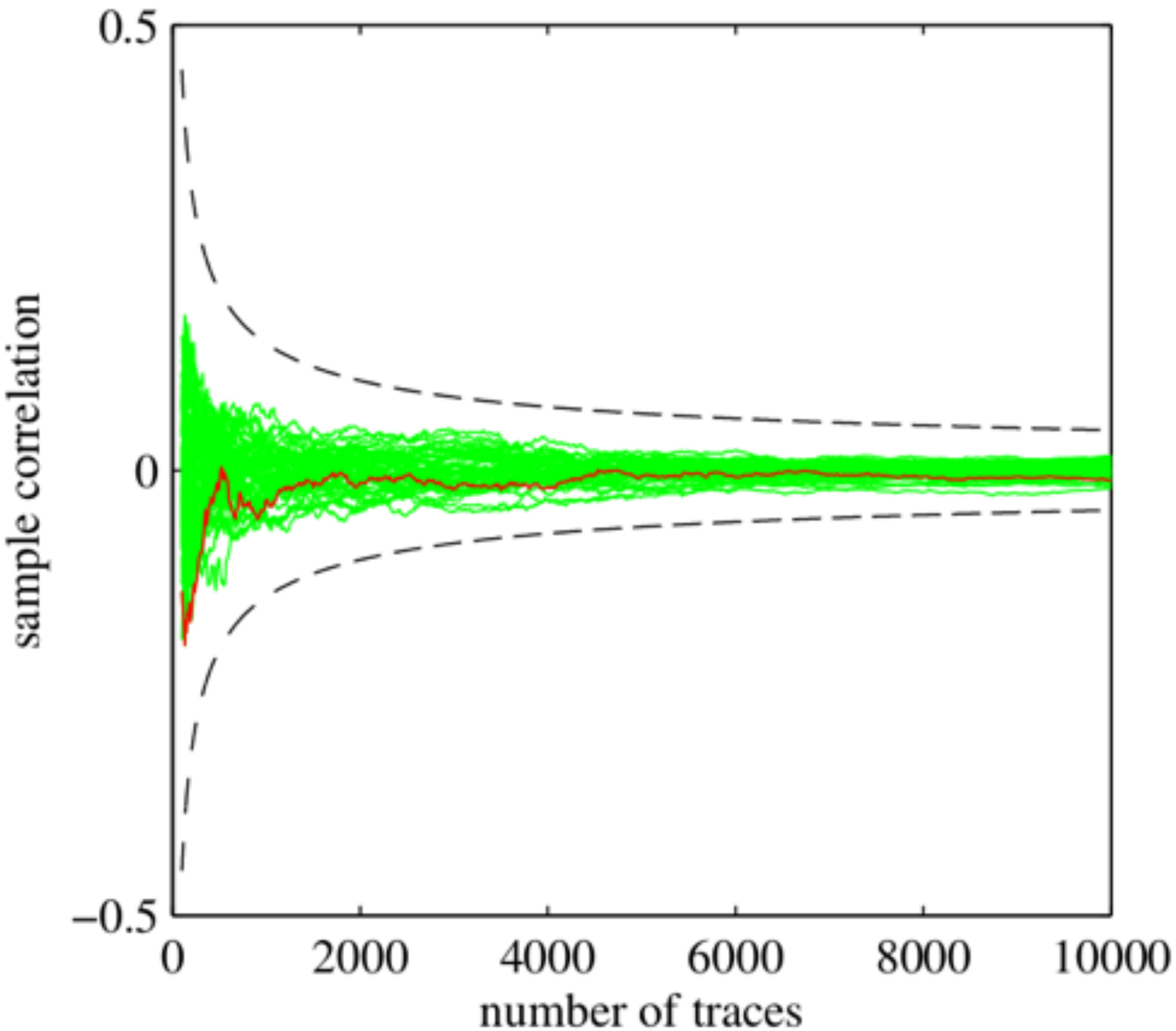

branch 1
branch 2
decoding
mean curve
sample ρ
$\rho = 0.25$, intermediate: $r'[0] \cdot c_1[0] + c_2[0]$
$\rho = 0.3$, intermediate: $r''[0] \cdot c_1[0]$
$\rho = 0.27$, intermediate: $a'[0]$
$\rho = 0.21$, intermediate: $m'[0]$
1000
2000
3000
4000
5000
6000
7000
time in cycles

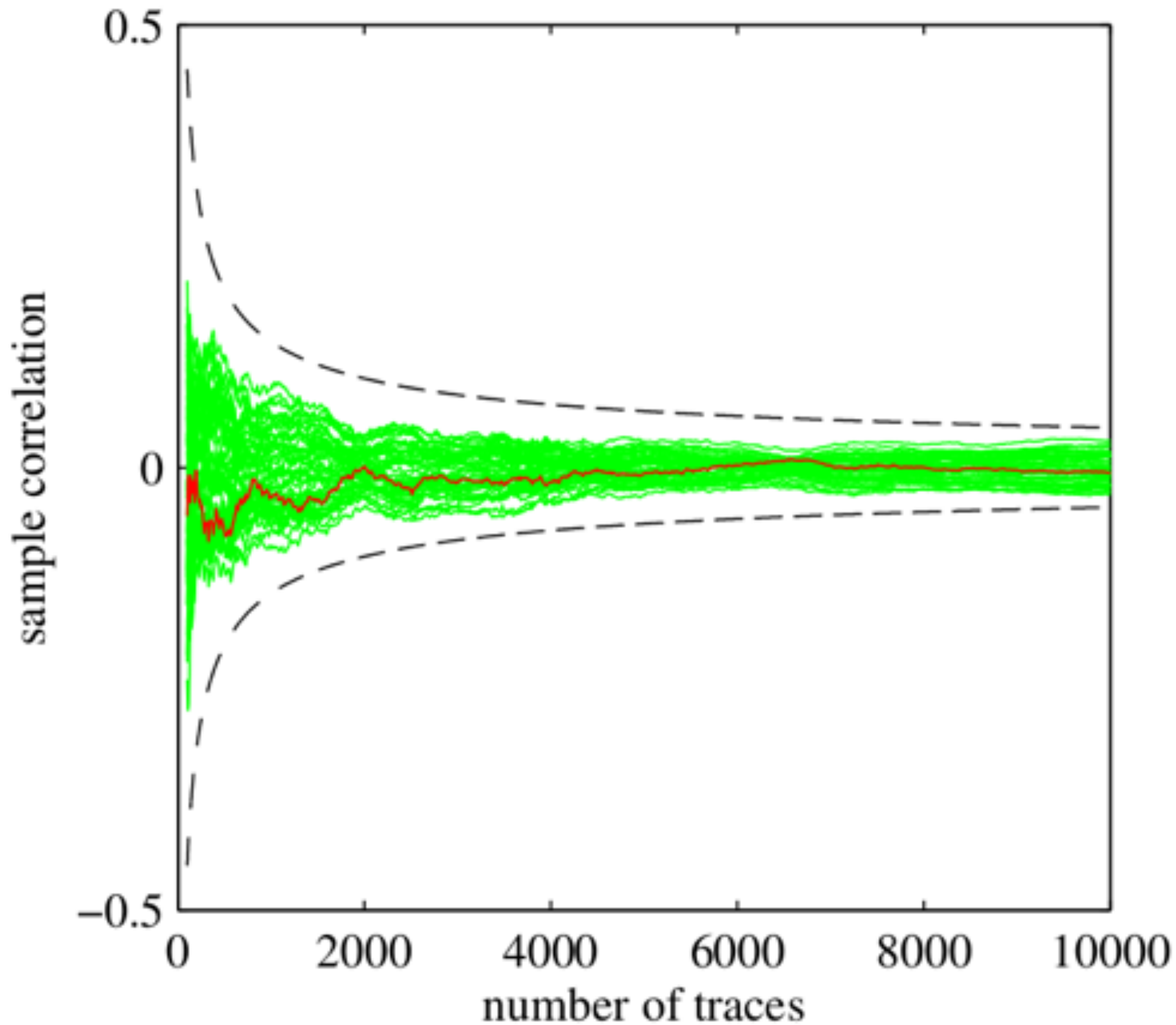
PRNG off



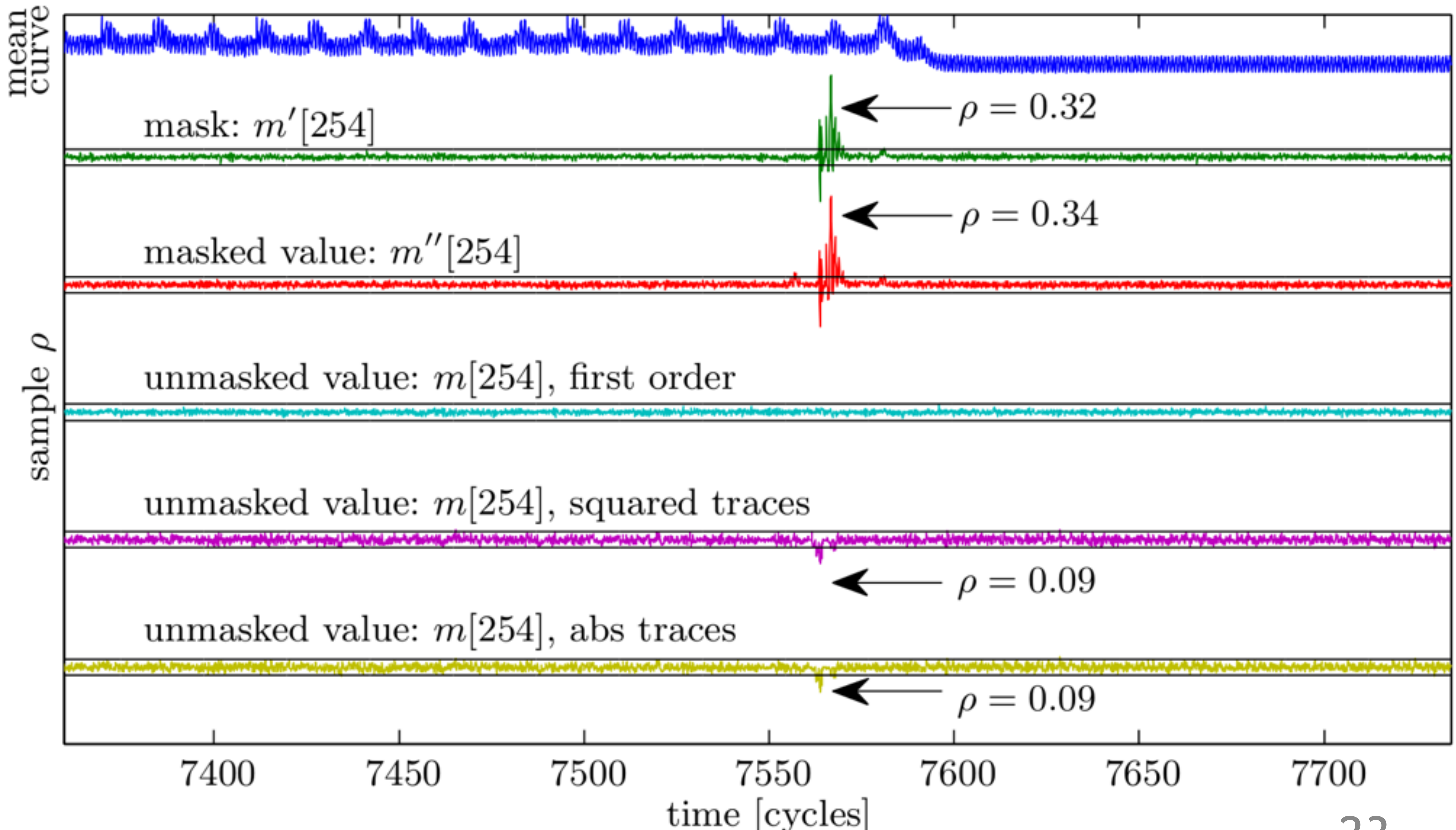
0.5
0
−0.5
sample correlation
0
2000
4000
6000
8000
10000
number of traces
0.5
0
−0.5
sample correlation
0
2000
4000
6000
8000
10000
number of traces

PRNG on

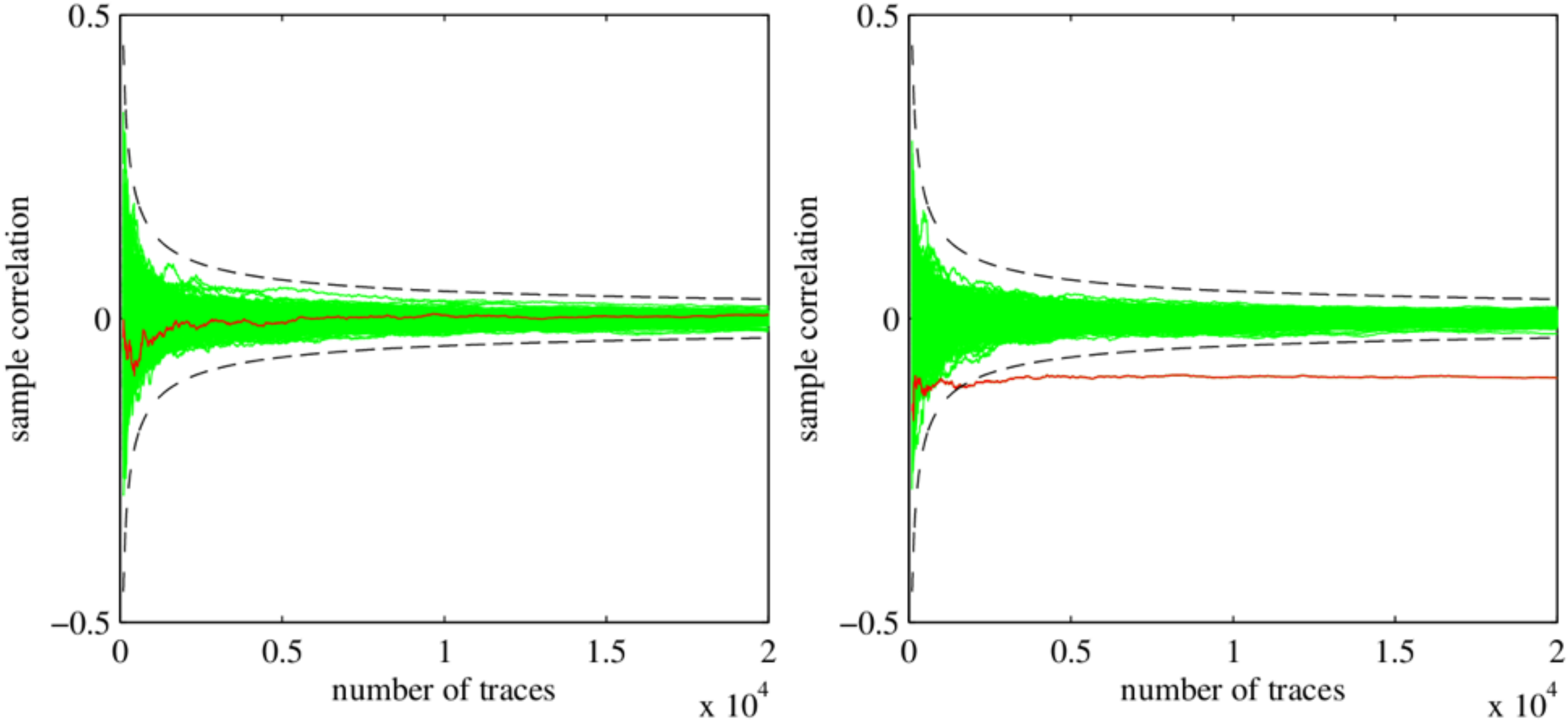

0.5
0
−0.5
sample correlation
0
2000
4000
6000
8000
10000
number of traces


0.5
0
−0.5
sample correlation
0
2000
4000
6000
8000
10000
number of traces

second order



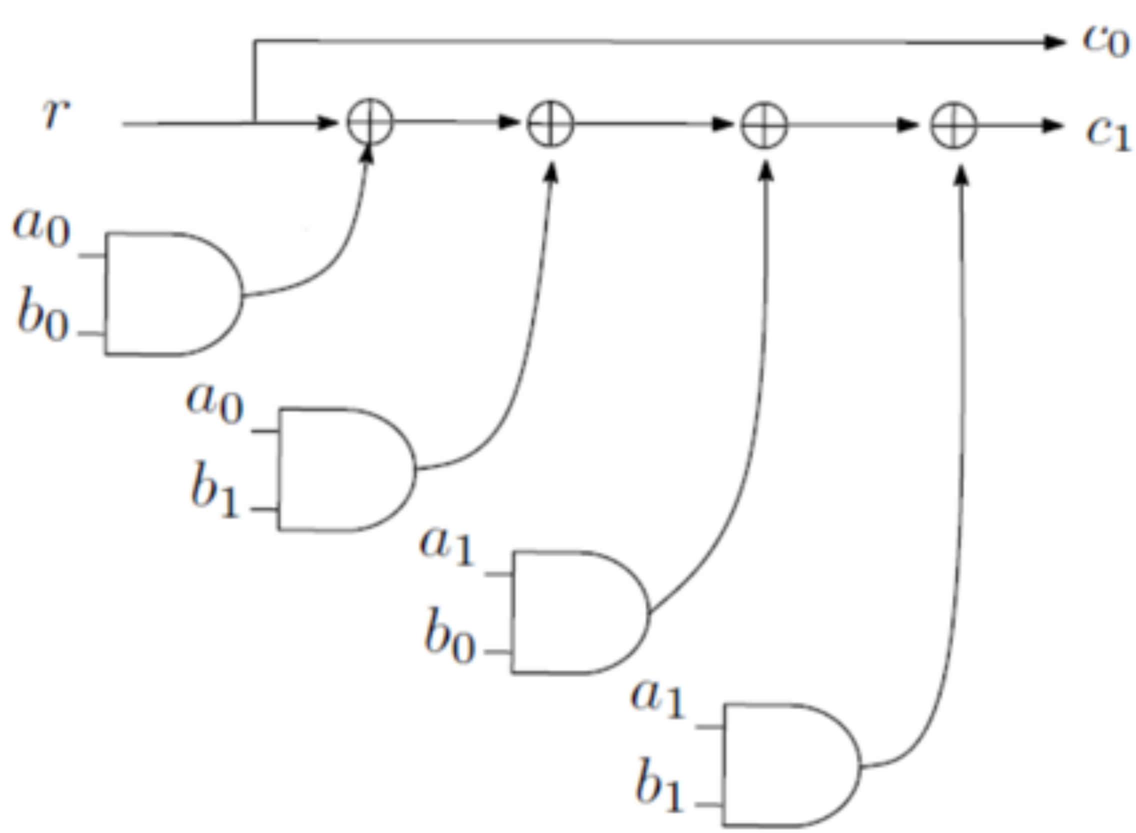
second order



ARM masked bitsliced implementation

Apply (hardware) gate-level masking
Substitute 5 macros with secure versions

- SXOR, SMOV, SROTL, SNOT: trivial
- SAND: Trichina gate




```
#define SAND(c, a, b){
  t0 = a[0] & b[0];
  t1 = a[0] & b[1];
  c[0] = RAND();
  t0 = t0 ^ c[0];
  t0 = t0 ^ t1;
  t1 = a[1] & b[0];
  t0 = t0 ^ t1;
  t1 = a[1] & b[1];
  t0 = t0 ^ t1;
  c[1] = t0;}
```

pqcrypto 2016 masking: split the inputs

pqcrypto 2016 masking: split the inputs

- ring-LWE decryption is additively homomorphic

pqcrypto 2016 masking: split the inputs

- ring-LWE decryption is additively homomorphic

$$\mathrm{decryption}(c_1, c_2) \oplus \mathrm{decryption}(c_1', c_2') = \mathrm{decryption}(c_1 + c_1', c_2 + c_2')$$

pqcrypto 2016 masking: split the inputs

- ring-LWE decryption is additively homomorphic

$$\mathrm{decryption}(c_1, c_2) \oplus \mathrm{decryption}(c_1', c_2') = \mathrm{decryption}(c_1 + c_1', c_2 + c_2')$$

- Procedure:

pqcrypto 2016 masking: split the inputs

- ring-LWE decryption is additively homomorphic

$$\text{decryption}(c_1, c_2) \oplus \text{decryption}(c'_1, c'_2) = \text{decryption}(c_1 + c'_1, c_2 + c'_2)$$

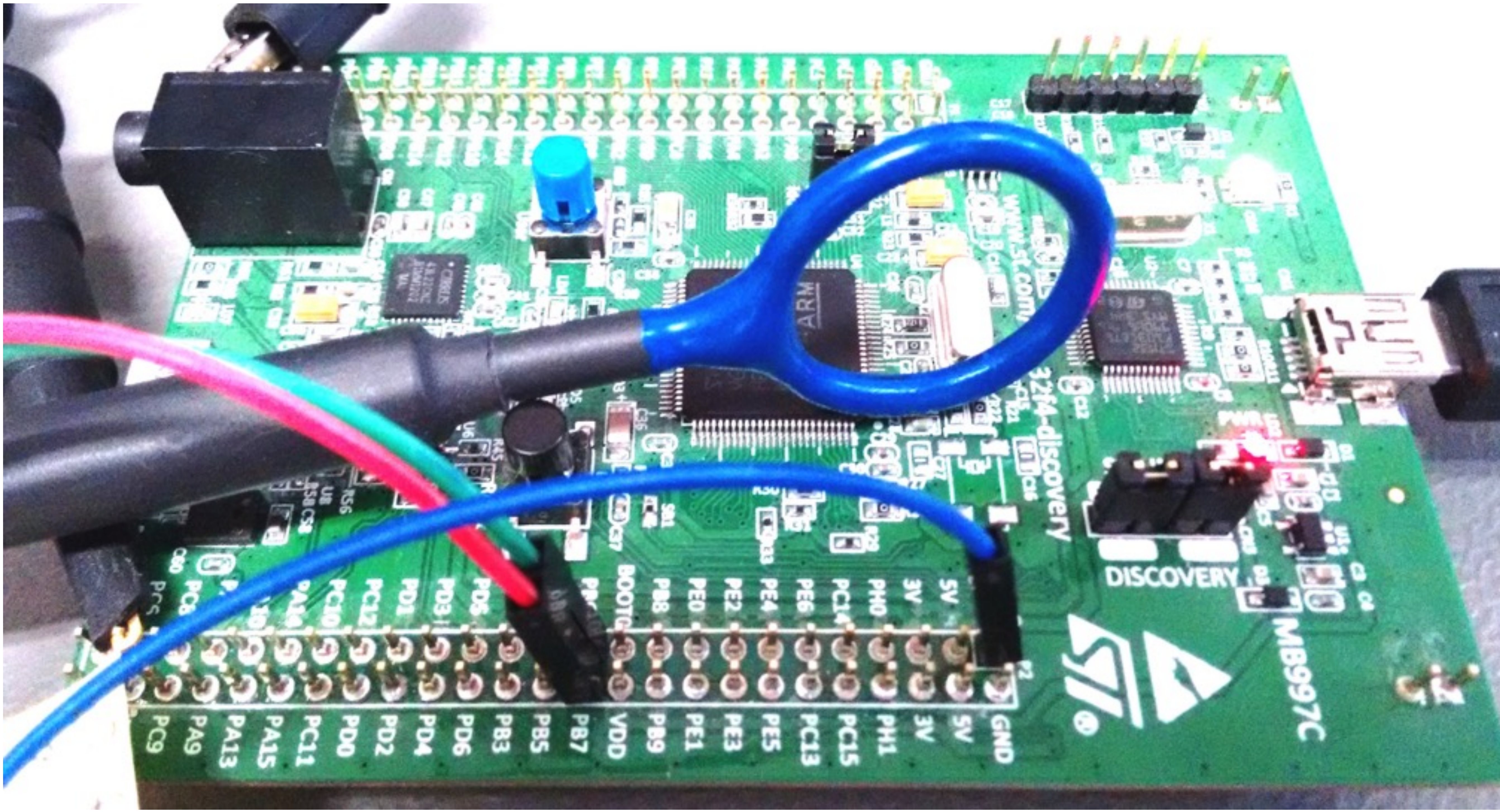
- Procedure:

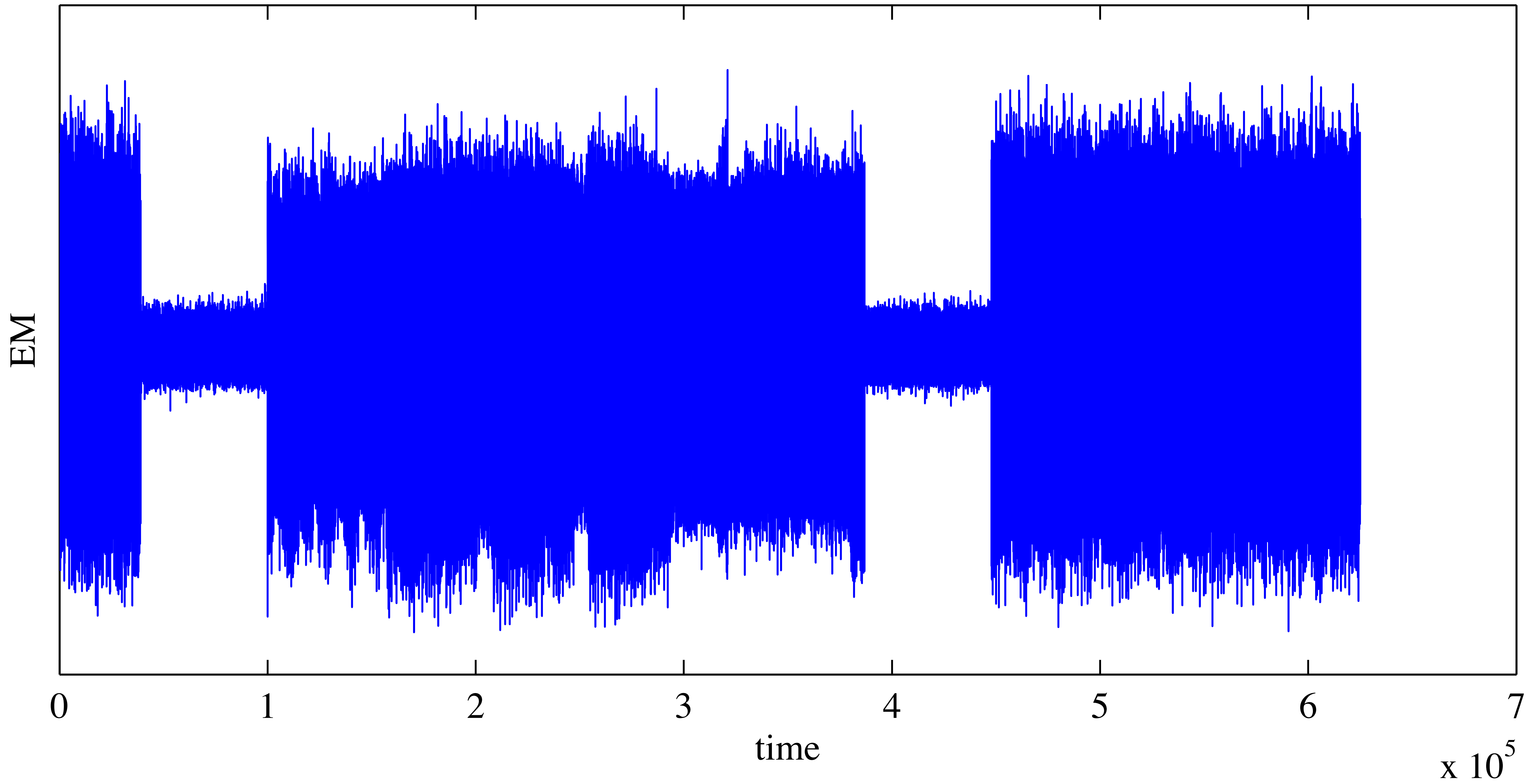
1. Internally generate a random message m' unknown to the adversary
2. Encrypt m' to (c'_1, c'_2)
3. Perform decryption$(c_1 + c'_1, c_2 + c'_2)$ to recover $m \oplus m'$.

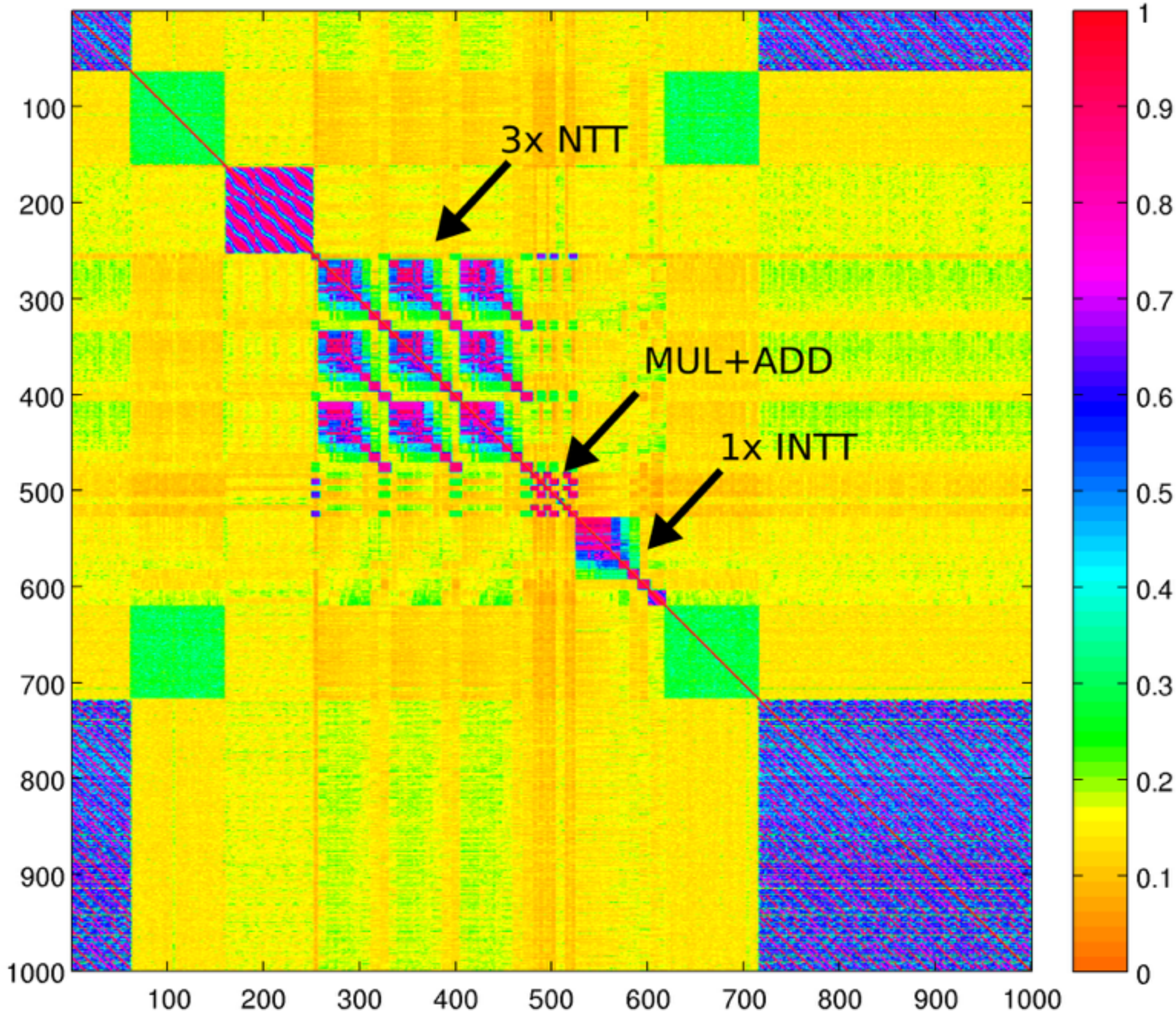
pqcrypto 2016 masking (2)

- Essentially ciphertext blinding
  - adversary looses control over input, can’t place predictions
- We don’t need a masked decoder! 😍
- Need to keep private + **public** keys on decryption
- Not provable secure

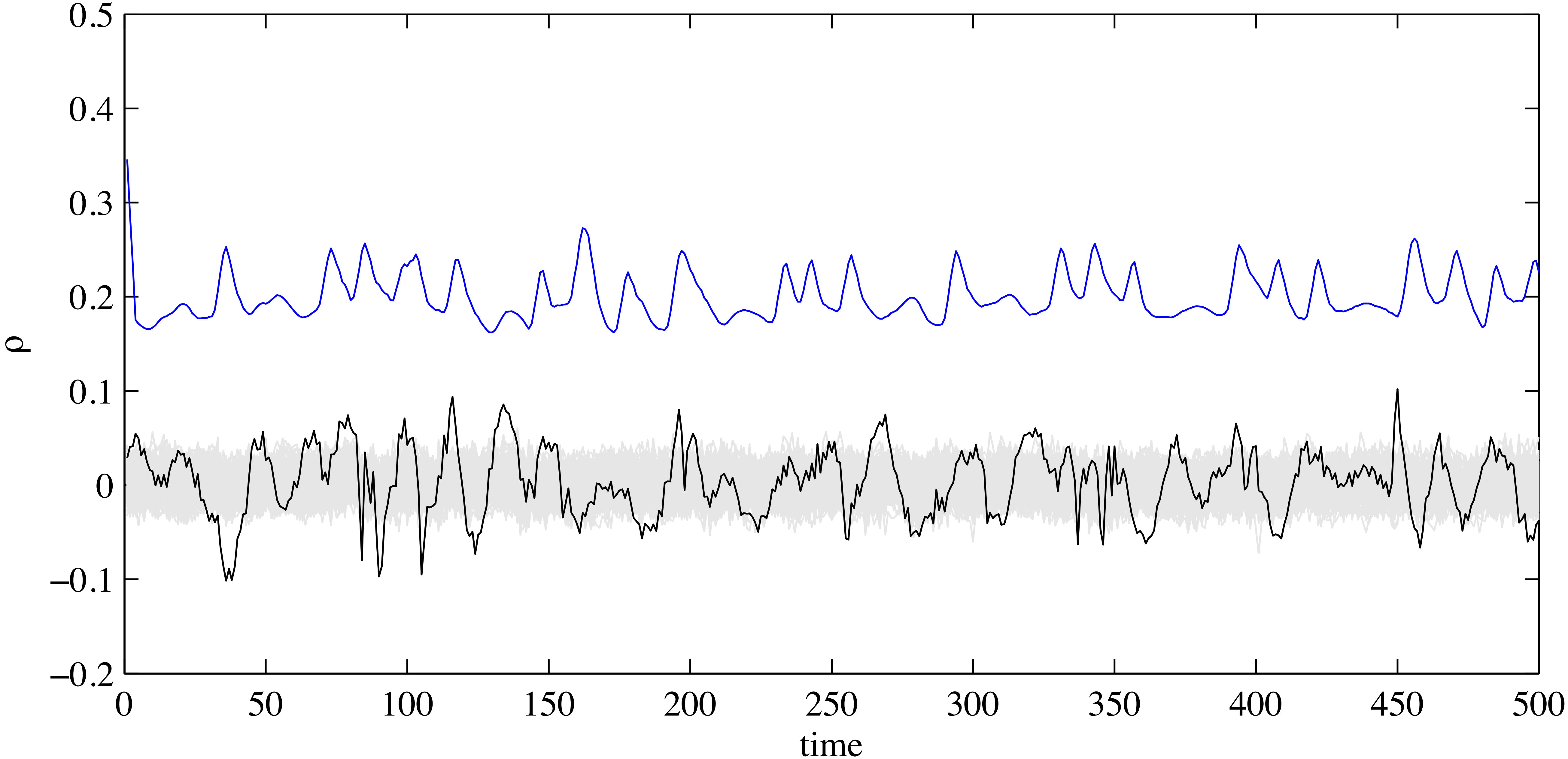
experiments

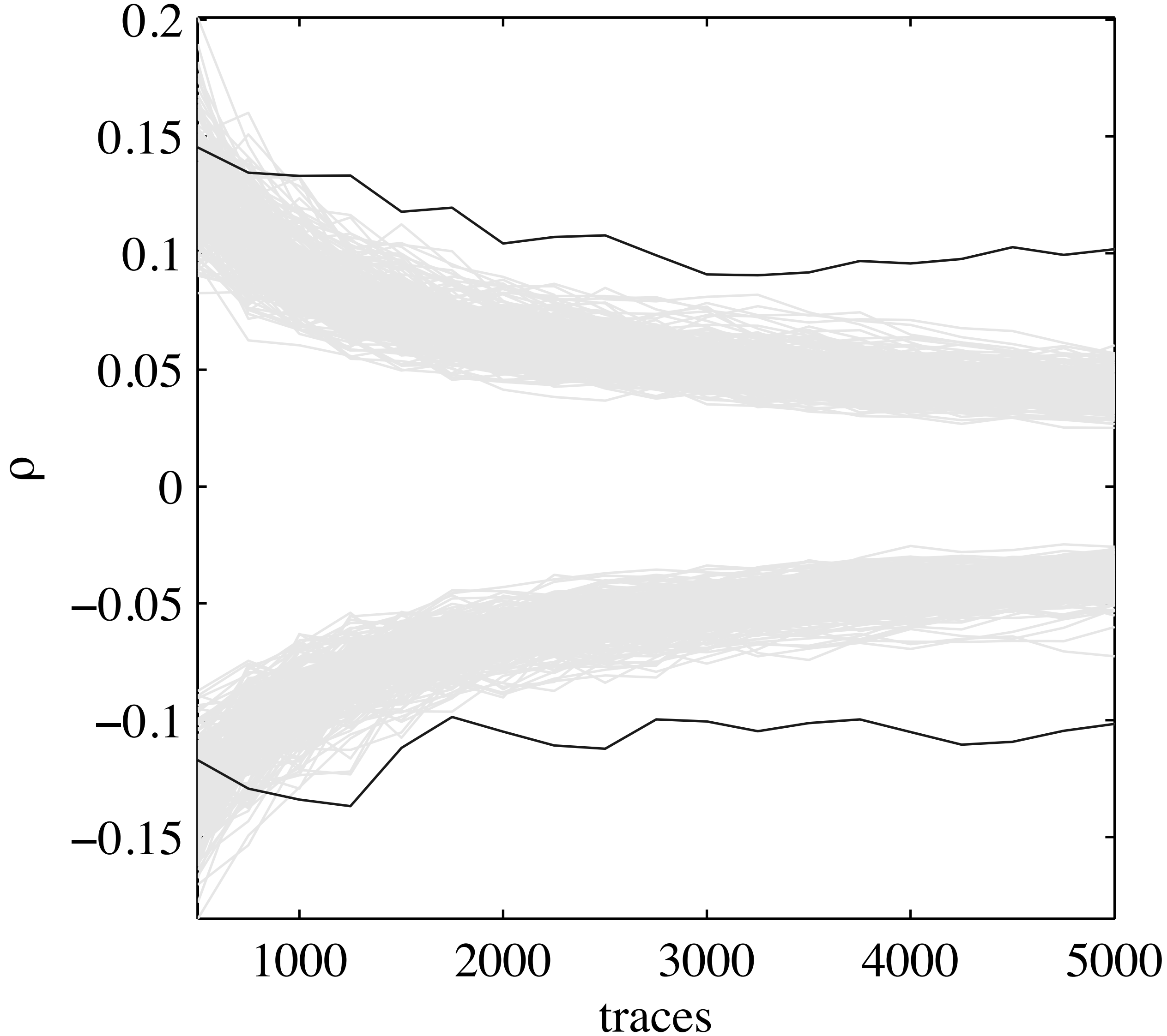


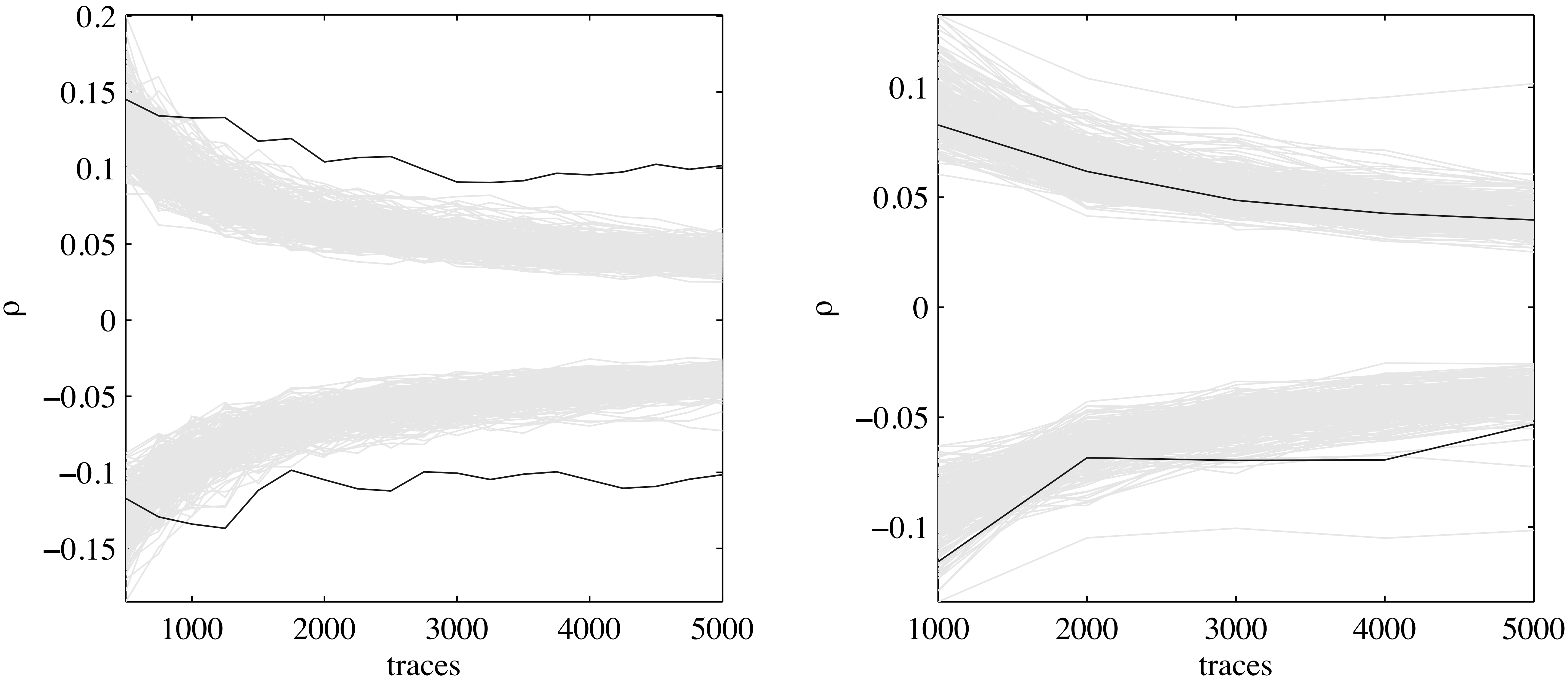

EM
0
1
2
3
4
5
6
7
time
x 10^5



3x NTT
MUL+ADD
1x INTT


0.5
0.4
0.3
0.2
0.1
0
−0.1
−0.2
ρ
0
50
100
150
200
250
300
350
400
450
500
time


0.2
0.15
0.1
0.05
0
−0.05
−0.1
−0.15
ρ
1000
2000
3000
4000
5000
traces


0.2
0.15
0.1
0.05
0
−0.05
−0.1
−0.15
ρ
1000
2000
3000
4000
5000
traces
0.1
0.05
0
−0.05
−0.1
ρ
1000
2000
3000
4000
5000
traces

comparision

	SCA security	overhead	implementation difficulty
unprotected	none	x1	med
CHES 2015	provable	x3	high
PQCrypto 2016	not provable	x4	med

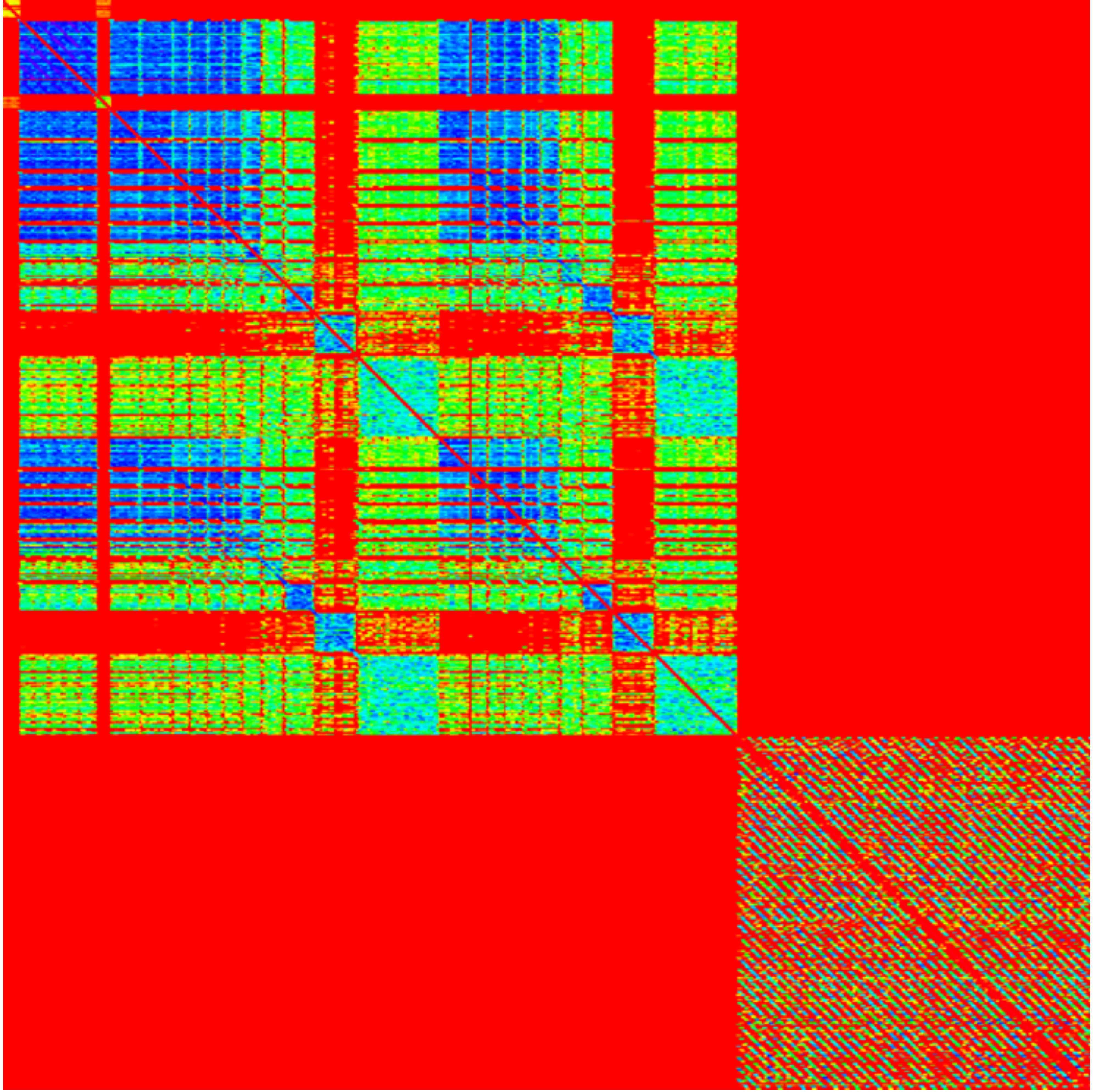
Conclusion

CHES 2015 (split key)

- Fully masked ring-LWE decryption
  - outputs Boolean shares
- Manageable overhead: x2.6 cycles wrt unprotected
- Small!
- Bespoke decoder
  - Error rate controlled
- Practical evaluation

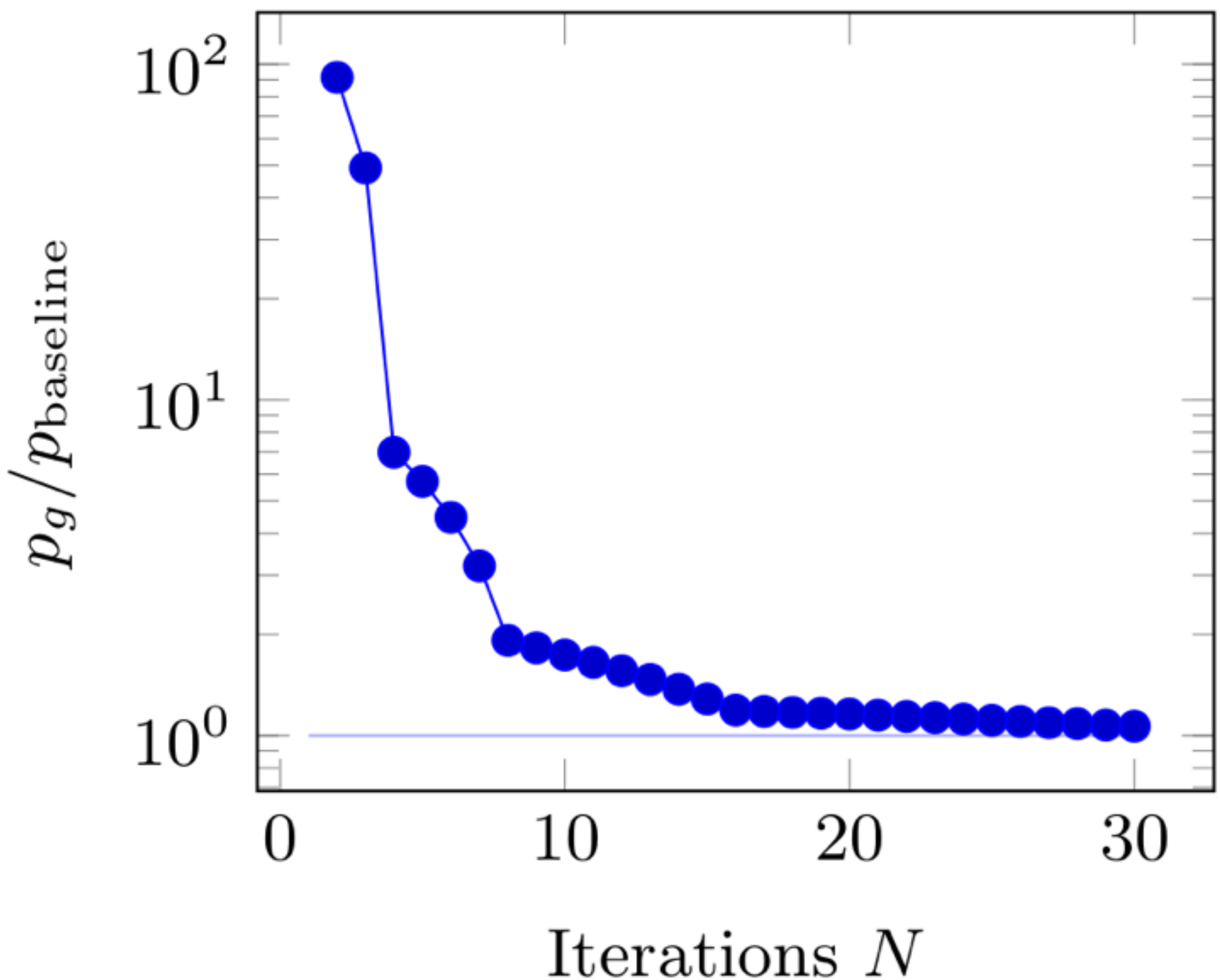
PQCrypto 2016 (split input)

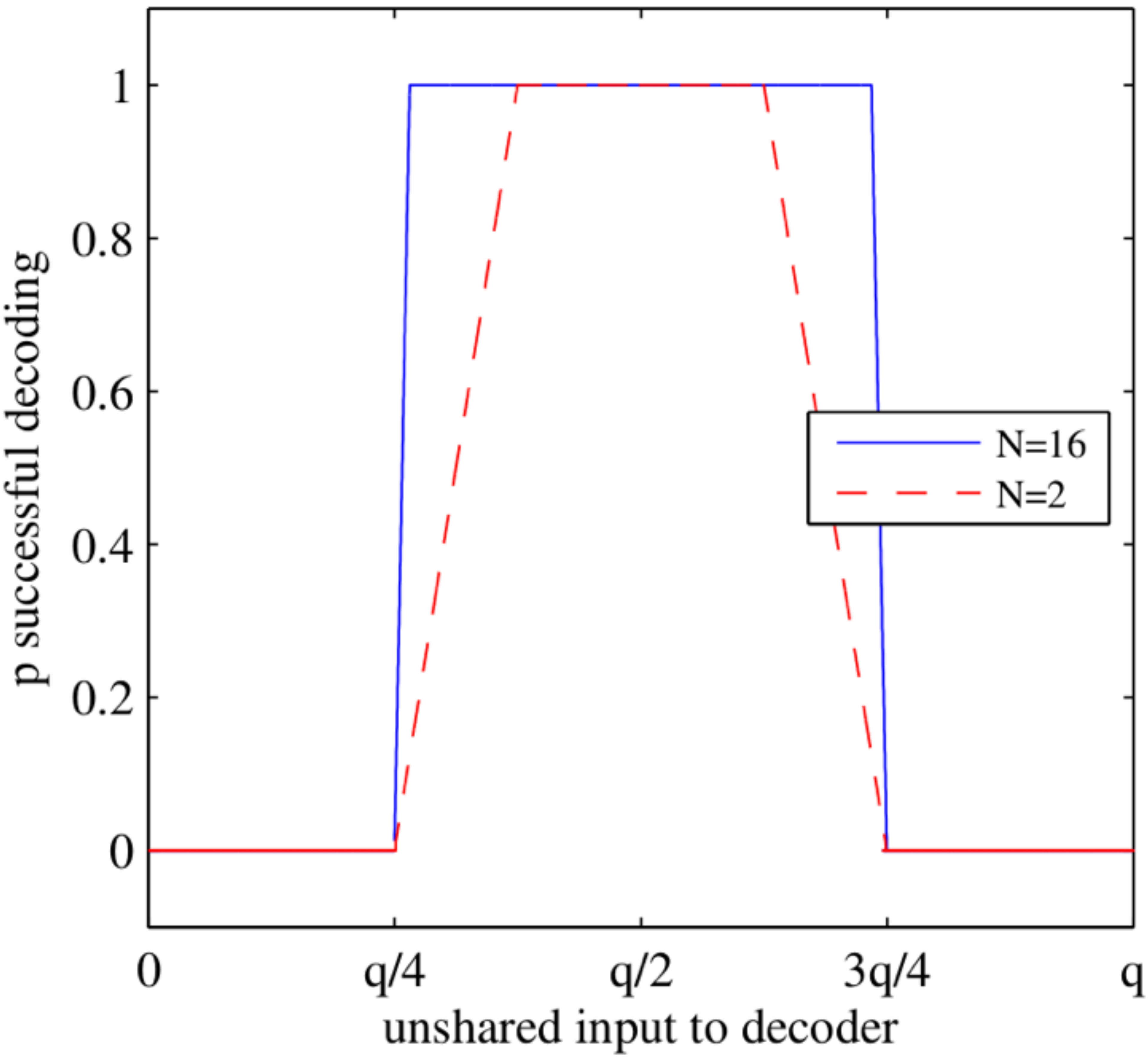
- Cheap masking
  - Not provable secure, but increases DPA resistance
  - Easy to implement: no new building blocks used, shares handled separately.
- Easy to upgrade from unmasked to masked
- Manageable overhead



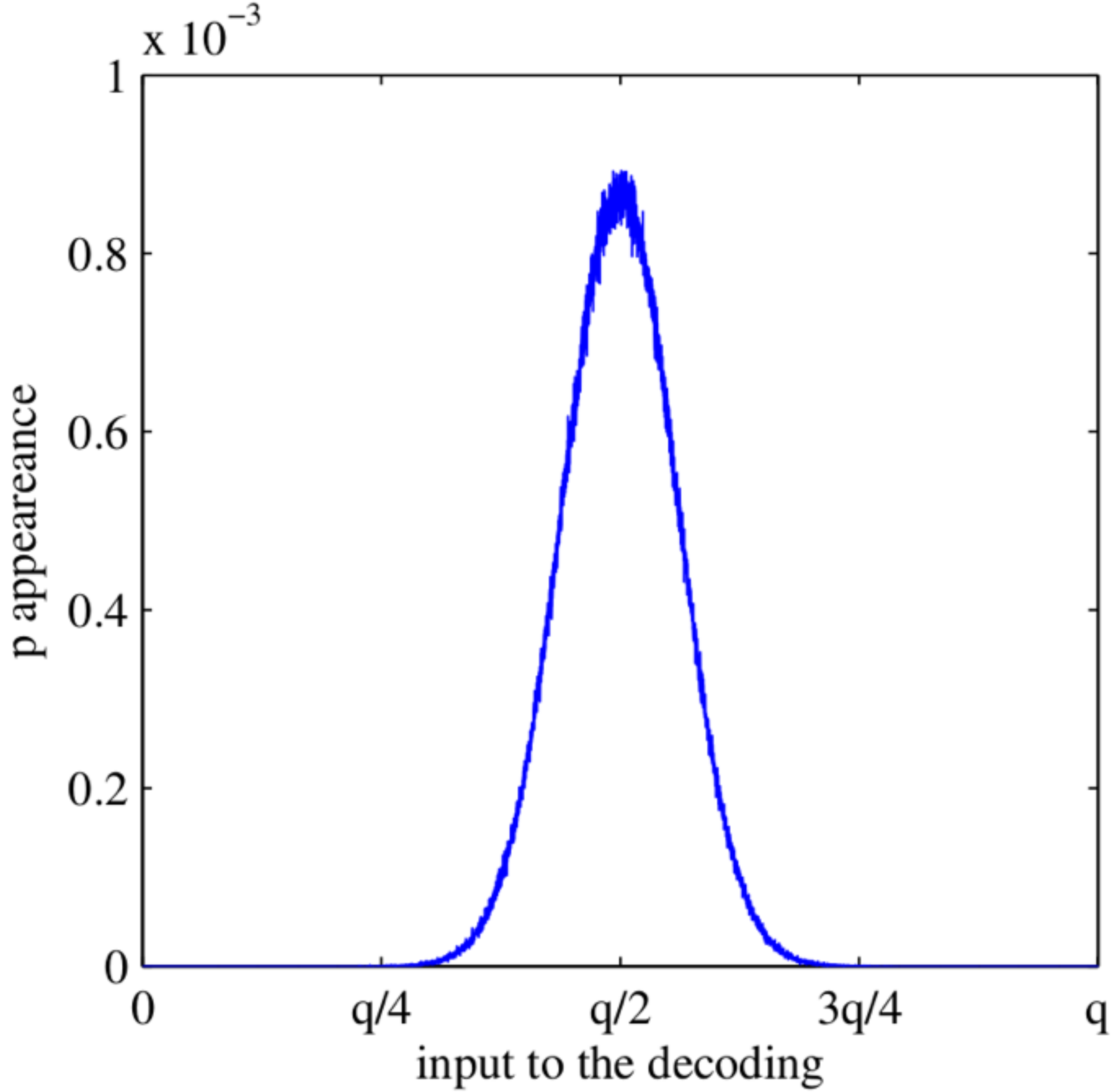
error rates

error rates





1
0.8
0.6
0.4
0.2
0
p successful decoding
N=16
N=2
0
q/4
q/2
3q/4
q
unshared input to decoder


x 10^-3
1
0.8
0.6
0.4
0.2
0
p appeareance
0
q/4
q/2
3q/4
q
input to the decoding