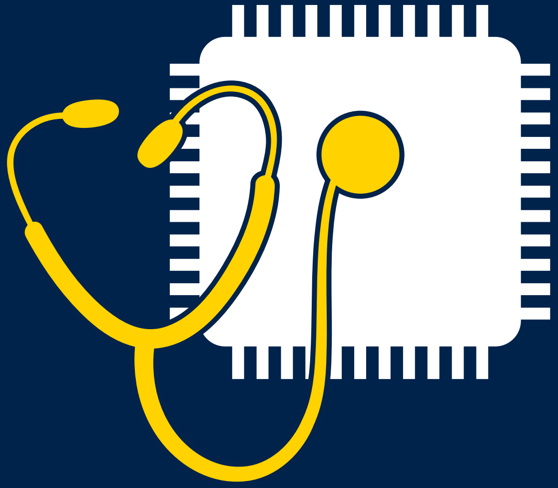
Improving Single-Trace Attacks on the Number-Theoretic Transform for Cortex-M4

Authors: **Guilhèm Assael** [1, 2], Philippe Elbaz-Vincent [2], Guillaume Reymond [1]

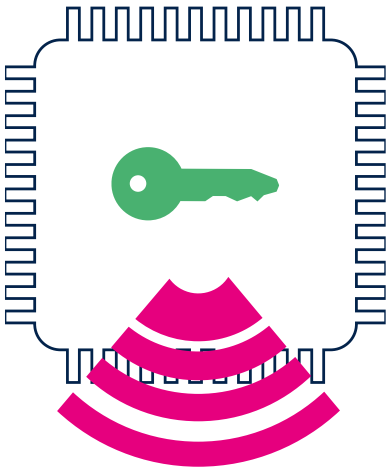
[1] STMicroelectronics Rousset, 13106 Rousset, France

[2] Univ. Grenoble Alpes, CNRS, IF, 38000 Grenoble, France

October 17, 2023



Purpose and summary



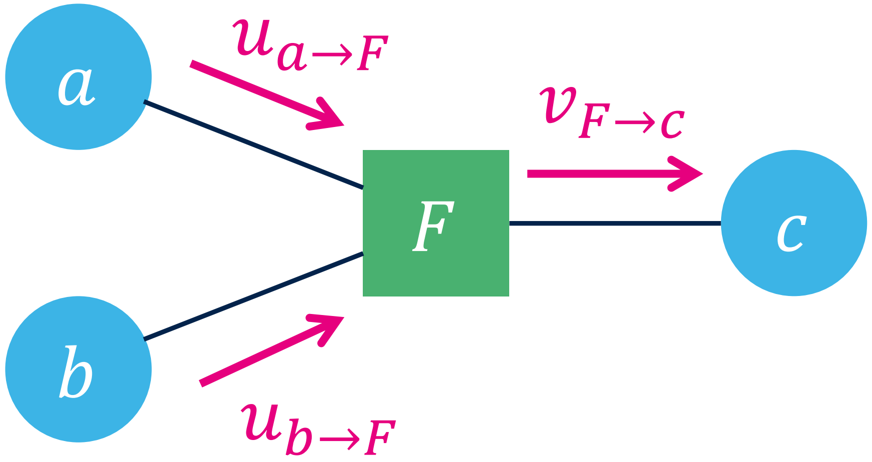
Side-channel attack: determine a **cryptographic secret** from the **environmental leakage** of an electronic device

- Vertical attack: observe several operations using the same secret, then combine the information from these executions
- **Horizontal / single-trace attack**: determine full or partial secret from a single execution of the algorithm

This work was published in [1] Assael, Elbaz-Vincent and Reymond, "Improving Single-Trace Attacks on the Number-Theoretic Transform for Cortex-M4," HOST 2023.

Belief-propagation (BP) attacks

- Optimization technique first applied to side-channel attacks in [2]
- Model the target algorithm as a **factor graph**
  - **Variable nodes**: unknown quantities
  - **Factor nodes**: relations between variables

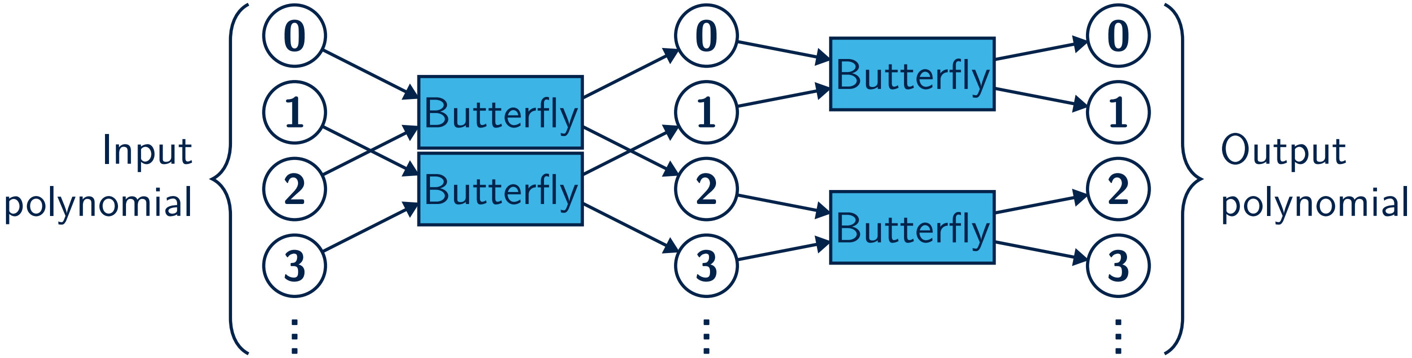


- Pass **messages** (→) between adjacent nodes
  - They represent the belief (estimated probability distribution) on a variable
  - **Variable nodes** compute their **output messages** based on their **input messages**
  - **Factor nodes**' computation is based on the **relation they represent** and their **input messages**
- Iterate message passing until convergence, then extract **marginal probabilities**

[2] Veyrat-Charvillon *et al.*, "Soft analytical side-channel attacks," in Advances in Cryptology – ASIACRYPT 2014.

Kyber NTT for Cortex-M4

- **Kyber** is a lattice-based key-encapsulation mechanism selected by NIST for PQC [3]
- It uses the **Number-Theoretic Transform** (NTT) for polynomial multiplication
  - Kyber NTT is made up of *butterfly* operations on pairs of coefficients, applied in 7 layers

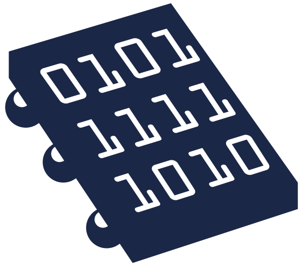


- Microcontrollers based on ARMv7E-M instruction set (*e.g.* Cortex-M4) have **DSP instructions** operating on **packed half-words.** They can implement Kyber NTT **two butterflies at a time** by packing pairs of coefficients into a 32-bit word [4]

[3] Alagic *et al.*, “Status report on the third round of the NIST post-quantum cryptography standardization process,” National Institute of Standards and Technology, 2022.
[4] Huang *et al.*, “Improved Plantard arithmetic for lattice-based cryptography,” TCHES 2022/4.

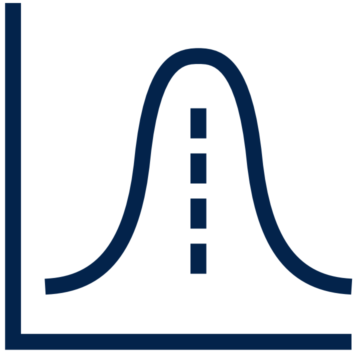
life.augmented

Leakage model



We assume that instructions **leak the result they write** to registers or RAM

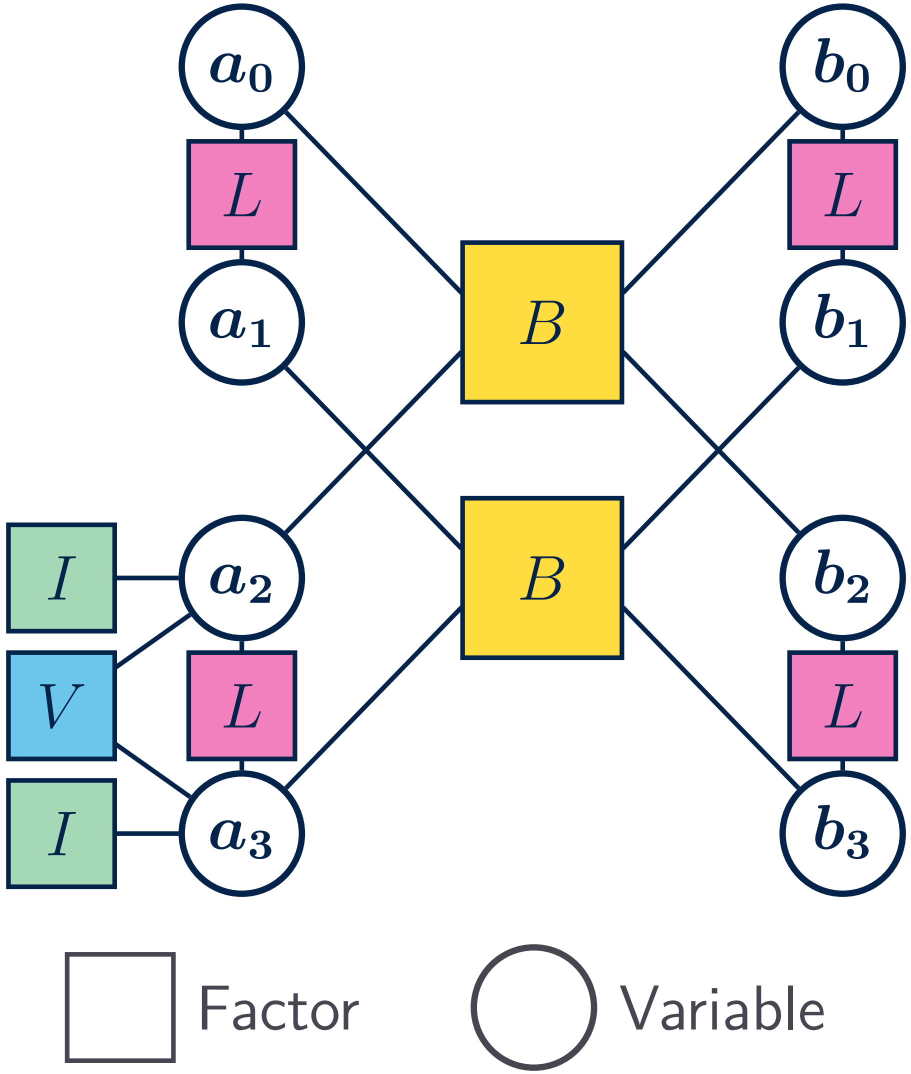
The leakage considered is **Hamming Weight** (between 0 and 32)



Exact Hamming-weight measurements are overlaid with centered **Gaussian noise** having configurable **standard deviation** σ_M

Factor graph for each double butterfly

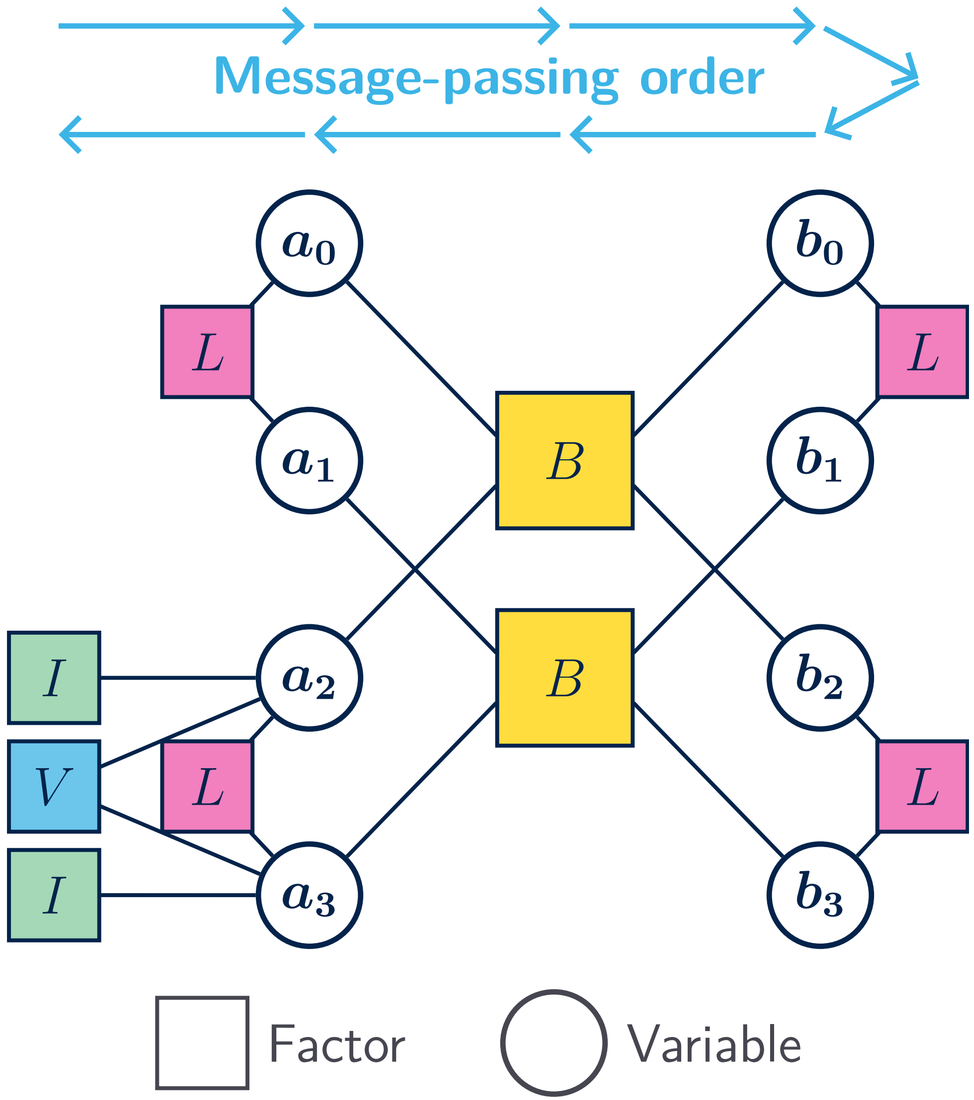
- We model the exact CPU instructions used
- Every 16-bit polynomial coefficient in each NTT layer is modeled by a variable node
- Types of factor nodes used
  - L leakage of load or store operations
  - I leakage from instructions on **single** coefficients
  - V leakage from instructions on **pairs** of coefficients
  - B butterfly equation



Message-passing order

- We use a ping-pong message schedule after [5]: start from the NTT input layer, propagate messages until last layer, and bounce back toward the input
- All same-type factors of each layer can be processed in parallel (up to 128 threads)

L	leakage of load or store operations
I	leakage from instructions on **single** coefficients
V	leakage from instructions on **pairs** of coefficients
B	butterfly equation



[5] Pessl *et al.*, "More practical single-trace attacks on the number theoretic transform," LATINCRYPT 2019.

Message damping and pruning

Pessl *et al*. [5] introduced **message damping** to reduce the risk of messages oscillating across iterations

We apply their technique with the following weighting:

- 95% weight for the message-update rule
- 5% weight for the old message value

Additionally, we perform **message pruning** by zeroing low-probability outcomes to speed up computations

[5] Pessl *et al*, “More practical single-trace attacks on the number theoretic transform,” LATINCRYPT 2019.

General setup

Environment

- Python 3.10, partially JIT-compiled with numba 0.55
- AMD EPYC 7713P @ 2 GHz, running on 32 cores

Target simulation

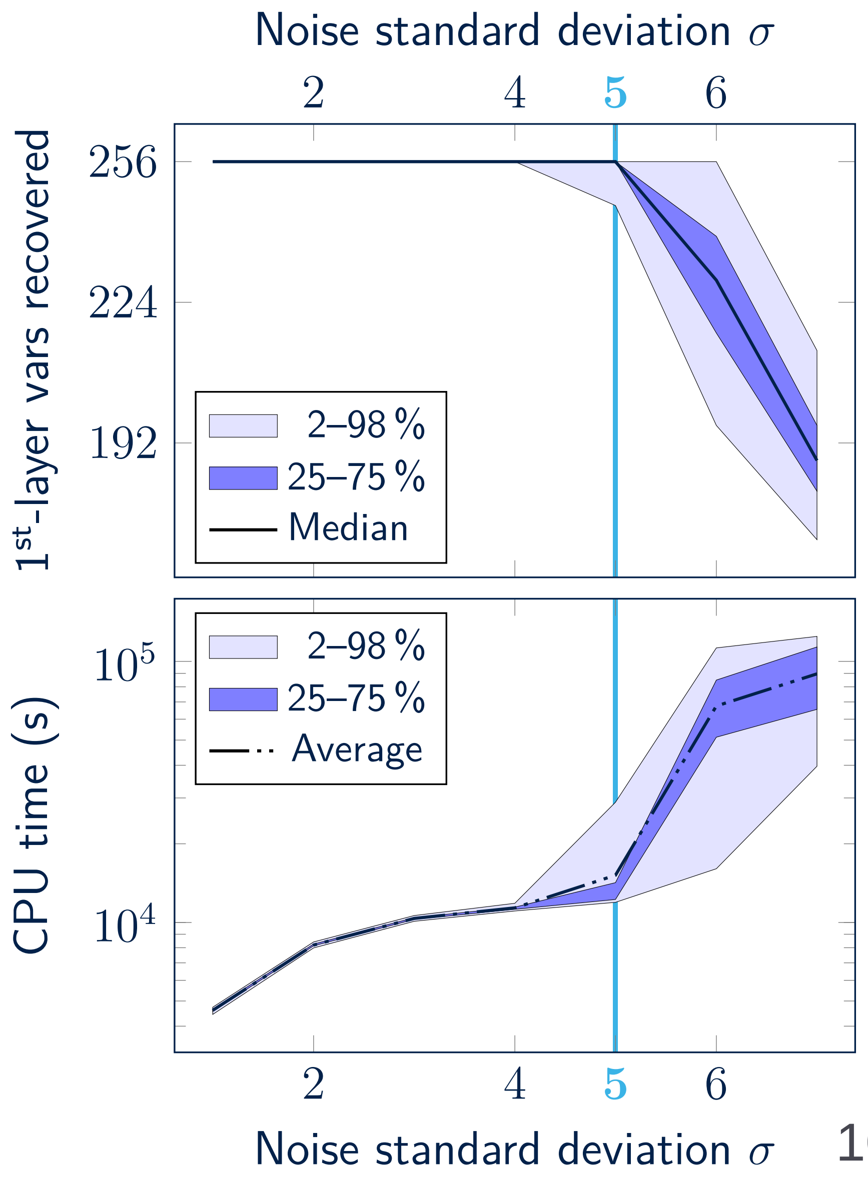
- Sample a **random input polynomial** with 256 coefficients
- Compute its NTT by **simulating the instructions** constituting the butterflies
- Measure the **Hamming weight** of all results written to registers and add **Gaussian noise**

Attack

- Build factor graph and configure factor nodes with the noised measurements
- Run the message passing until **convergence**, **iteration limit** or **failure**
- Compute the marginals of input-layer variables and keep the **highest-probability outcome** of each

Results for binomially-distributed NTT input

- The input polynomial is sampled with coefficients **binomially distributed** in $[-3, 3]$
- The standard deviation σ of measurement noise is **known**
- More than **75%** of the trials reach **perfect success** (all coefficients recovered) up to $\boldsymbol{\sigma = 5}$
- Average CPU time ≤ 3 hours up to $\sigma = 4$

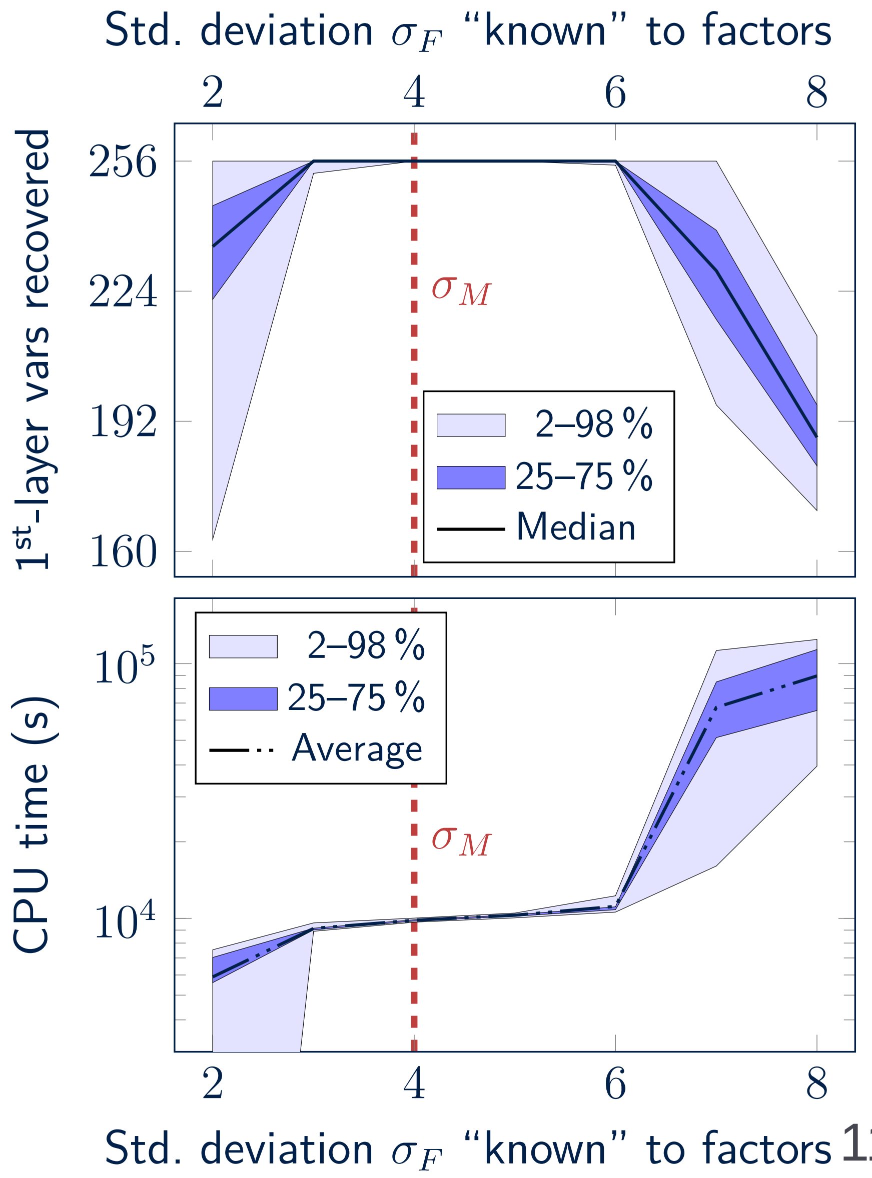


Reality check: what if noise level is unknown?

- The input polynomial is sampled with coefficients **binomially distributed** in $[-3,3]$
- The standard deviation σ_M of measurement noise is **unknown** and **approximated by σ_F**
- **Actual** standard deviation is $\sigma_M = 4$

- More than **90%** of the trials reach **perfect success** (all coefficients recovered) when $\sigma_F \in [3,6]$

- Average CPU time $\leq$ 3 hours for $\sigma_F \in [3,6]$



life.augmented

Attack exploitation

Threat model: attacker can record side-channel traces and may request the decapsulation of chosen ciphertexts

Message recovery

- Used to recover the encapsulated shared secret
- Recover $\boldsymbol{r}$ during encryption

Key recovery

- Recover $\boldsymbol{s}$ or $\boldsymbol{e}$ during key generation
- Or perform **message recovery** during the decapsulation of chosen ciphertexts, and use as a decryption-failure oracle [6]

Kyber encryption (simplified)

Input: Public key $pk = (\hat{\mathbf{A}}, \hat{\mathbf{t}}) \in R_q^{k\times k} \times R_q^k$
Input: Message $m \in \{0,1\}^n$
Output: Ciphertext $c = (\mathbf{u}, v) \in R_q^k \times R_q$

$\mathbf{r} \leftarrow \mathcal{B}_{\eta_1}^k \qquad \mathbf{e_1} \leftarrow \mathcal{B}_{\eta_1}^k$ // *Binomial sampling*
$e_2 \leftarrow \mathcal{B}_{\eta_2}$ // *Binomial sampling*
$\hat{\mathbf{r}} = \mathrm{NTT}(\mathbf{r})$
$\mathbf{u} = \mathrm{NTT}^{-1}(\hat{\mathbf{A}}^\top \circ \hat{\mathbf{r}}) + \mathbf{e_1}$
$v = \mathrm{NTT}^{-1}(\hat{\mathbf{t}}^\top \circ \hat{\mathbf{r}}) + e_2 + \lceil q/2 \rfloor m$
return $c = (\mathbf{u}, v)$

Kyber key generation (simplified)

Output: Private key $sk = \hat{\mathbf{s}} \in R_q^k$
Output: Public key $pk = (\hat{\mathbf{A}}, \hat{\mathbf{t}}) \in R_q^{k\times k} \times R_q^k$

$\hat{\mathbf{A}} \leftarrow R_q^{k\times k}$ // *Uniform sampling*
$\mathbf{s} \leftarrow \mathcal{B}_{\eta_1}^k \qquad \mathbf{e} \leftarrow \mathcal{B}_{\eta_1}^k$ // *Binomial sampling*
$\hat{\mathbf{s}} = \mathrm{NTT}(\mathbf{s}) \qquad \hat{\mathbf{e}} = \mathrm{NTT}(\mathbf{e})$
$\hat{\mathbf{t}} = \hat{\mathbf{A}} \circ \hat{\mathbf{s}} + \hat{\mathbf{e}}$
return $sk = \hat{\mathbf{s}},\ pk = (\hat{\mathbf{A}}, \hat{\mathbf{t}})$

[6] Hermelink *et al.*, “Fault-Enabled Chosen-Ciphertext Attacks on Kyber,” INDOCRYPT 2021.

life.augmented

Impact of countermeasures?

✕ Masking

- Not effective as the attacker can recover shares independently and combine them after the attack, or during the attack given modifications to the graph [5]
- **Masking only reduces noise tolerance**

☑ Shuffling

- **Very effective** [5]: **decorrelates timing from the corresponding data**
- Some adaptations of the attack have been proposed [7]

[5] Pessl *et al.*, “More practical single-trace attacks on the number theoretic transform,” LATINCRYPT 2019.
[7] Hermelink *et al.*, “Adapting belief propagation to counter shuffling of NTTs,” TCHES 2023/1.

Summarizing our contribution

We adapt the **belief-propagation attack** to an **optimized Cortex-M4 implementation** of Kyber

We show that **accurate modelling** of the algorithm allows the attack to **tolerate high noise**, up to standard deviation $\sigma = 5$ (for Hamming-weight measurements between 0 and 32)

We highlight that the **attack performs well** when the amplitude of **measurement noise is not precisely known**

References

[1] G. Assael, P. Elbaz-Vincent and G. Reymond, "Improving Single-Trace Attacks on the Number-Theoretic Transform for Cortex-M4," 2023 IEEE International Symposium on Hardware Oriented Security and Trust (HOST), San Jose, CA, USA, 2023, pp. 111-121, doi: 10.1109/HOST55118.2023.10133270.

[2] N. Veyrat-Charvillon, B. Gérard, and F.-X. Standaert, “Soft analytical side-channel attacks,” in Advances in Cryptology – ASIACRYPT 2014, P. Sarkar and T. Iwata, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 282–296.

[3] G. Alagic, D. Apon, D. Cooper, Q. Dang, T. Dang, J. Kelsey, J. Lichtinger, Y.-K. Liu, C. Miller, D. Moody, R. Peralta, R. Perlner, A. Robinson, and D. Smith-Tone, “Status report on the third round of the NIST post-quantum cryptography standardization process,” National Institute of Standards and Technology, 2022.

[4] J. Huang, J. Zhang, H. Zhao, Z. Liu, R. C. C. Cheung, Ç. K. Koç, and D. Chen, “Improved Plantard arithmetic for lattice-based cryptography,” IACR Transactions on Cryptographic Hardware and Embedded Systems, vol. 2022, no. 4, p. 614–636, Aug. 2022.

[5] P. Pessl and R. Primas, “More practical single-trace attacks on the number theoretic transform,” in Progress in Cryptology – LATINCRYPT 2019, P. Schwabe and N. Thériault, Eds. Cham: Springer International Publishing, 2019, pp. 130–149.

[6] J. Hermelink, P. Pessl, T. Pöppelmann, “Fault-Enabled Chosen-Ciphertext Attacks on Kyber,” Progress in Cryptology – INDOCRYPT 2021, INDOCRYPT 2021, P. Schwabe and N. Thériault, Eds. Cham: Springer International Publishing, 2021, pp. 311–334.

[7] J. Hermelink, S. Streit, E. Strieder, and K. Thieme, “Adapting belief propagation to counter shuffling of NTTs,” IACR Transactions on Cryptographic Hardware and Embedded Systems, vol. 2023, no. 1, pp. 60–88, Nov. 2022.

life.augmented

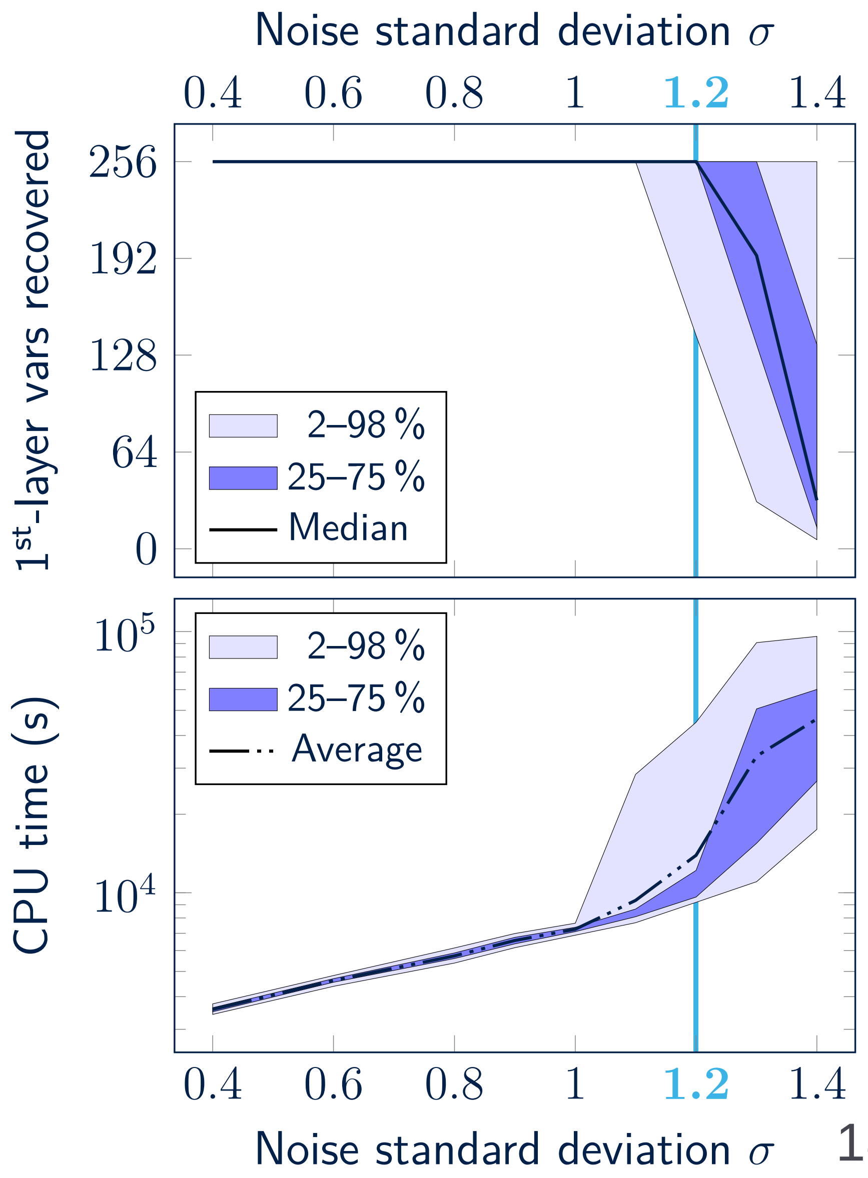
Thank you

life.augmented

Appendices

Results for uniformly-distributed NTT input

- The input polynomial is sampled with coefficients **uniformly distributed** in $[-\lceil q/2 \rceil, \lfloor q/2 \rfloor]$ ($q = 3329$)
- The standard deviation σ of measurement noise is **known**



- More than **90%** of the trials reach **perfect success** (all coefficients recovered) up to $\boldsymbol{\sigma = 1.2}$
- Average CPU time $\leq$ 3 hours up to $\sigma = 1.1$

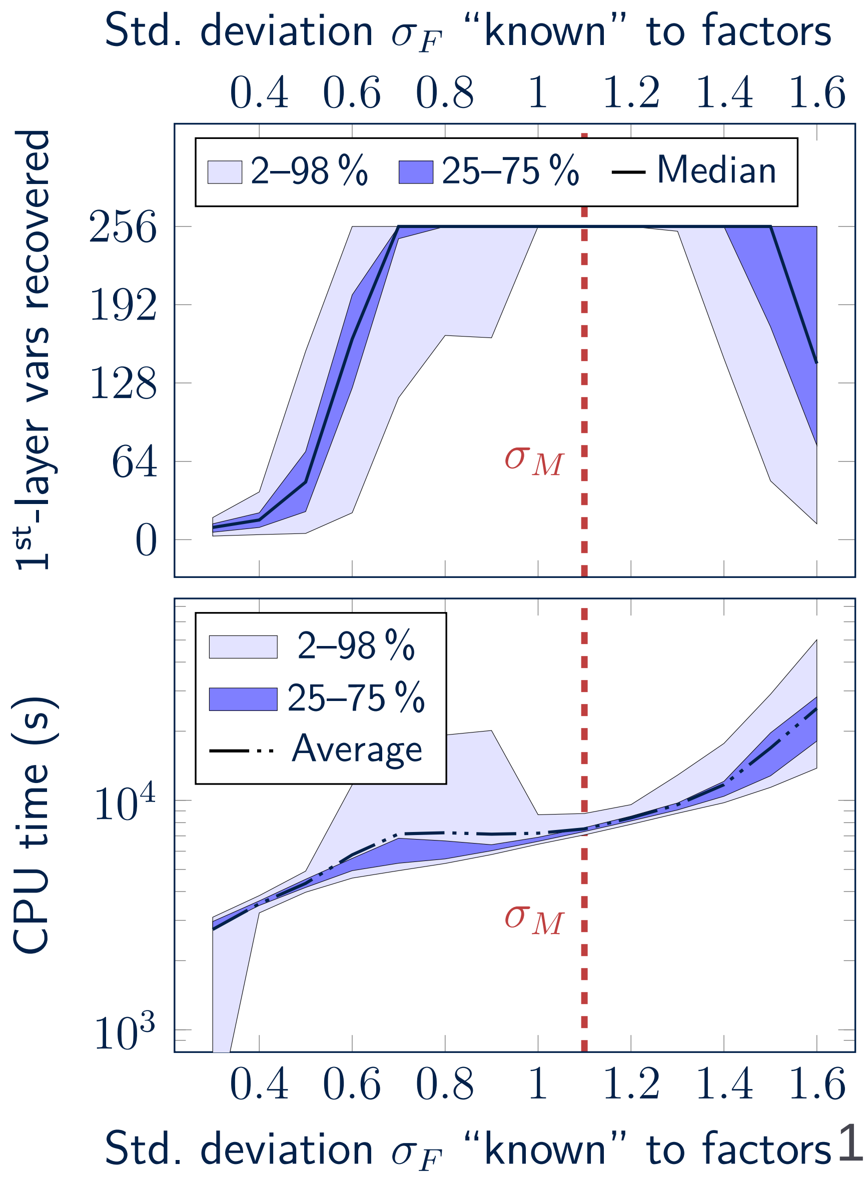
life.augmented

Uniformly-distributed NTT input and unknown noise

- The input polynomial is sampled with coefficients **uniformly distributed** in $[-\lceil q/2 \rceil, \lfloor q/2 \rfloor]$ ($q = 3329$)
- The standard deviation σ_M of measurement noise is **unknown** and **approximated by** σ_F
- **Actual** standard deviation is $\sigma_M = 1.1$

- More than **75%** of the trials reach perfect success (all coefficients recovered) when $\boldsymbol{\sigma_F \in [0.8, 1.4]}$

- Average CPU time $\leq$ 3 hours for $\sigma_F \in [0.8, 1.4]$



life.augmented

Possible attack paths on IND-CPA Kyber

Kyber key generation (simplified)

Output: Private key $sk = \hat{\mathbf{s}} \in R_q^k$
Output: Public key $pk = (\hat{\mathbf{A}}, \hat{\mathbf{t}}) \in R_q^{k \times k} \times R_q^k$

$\hat{\mathbf{A}} \leftarrow R_q^{k \times k}$ // *Uniform sampling*
$\mathbf{s} \leftarrow \mathcal{B}_{\eta_1}^k \quad \mathbf{e} \leftarrow \mathcal{B}_{\eta_1}^k$ // *Binomial sampling*
$\hat{\mathbf{s}} = \mathrm{NTT}(\mathbf{s}) \quad \hat{\mathbf{e}} = \mathrm{NTT}(\mathbf{e})$
$\hat{\mathbf{t}} = \hat{\mathbf{A}} \circ \hat{\mathbf{s}} + \hat{\mathbf{e}}$
return $sk = \hat{\mathbf{s}}$, $pk = (\hat{\mathbf{A}}, \hat{\mathbf{t}})$

Kyber decryption (simplified)

Input: Private key $sk = \hat{\mathbf{s}} \in R_q^k$
Input: Ciphertext $c = (\mathbf{u}, v) \in R_q^k \times R_q$
Output: Message $m \in \{0,1\}^n$

$w = v - \mathrm{NTT}^{-1}(\hat{\mathbf{s}}^{\mathsf{T}} \circ \mathrm{NTT}(\mathbf{u}))$
$m = \lceil (2/q) w \rfloor \bmod 2$
return m

Kyber encryption (simplified)

Input: Public key $pk = (\hat{\mathbf{A}}, \hat{\mathbf{t}}) \in R_q^{k \times k} \times R_q^k$
Input: Message $m \in \{0,1\}^n$
Output: Ciphertext $c = (\mathbf{u}, v) \in R_q^k \times R_q$

$\mathbf{r} \leftarrow \mathcal{B}_{\eta_1}^k \quad \mathbf{e_1} \leftarrow \mathcal{B}_{\eta_1}^k$ // *Binomial sampling*
$e_2 \leftarrow \mathcal{B}_{\eta_2}$ // *Binomial sampling*
$\hat{\mathbf{r}} = \mathrm{NTT}(\mathbf{r})$
$\mathbf{u} = \mathrm{NTT}^{-1}(\hat{\mathbf{A}}^{\mathsf{T}} \circ \hat{\mathbf{r}}) + \mathbf{e_1}$
$v = \mathrm{NTT}^{-1}(\hat{\mathbf{t}}^{\mathsf{T}} \circ \hat{\mathbf{r}}) + e_2 + \lceil q/2 \rfloor m$
return $c = (\mathbf{u}, v)$

Key-recovery attack

Message-recovery attack

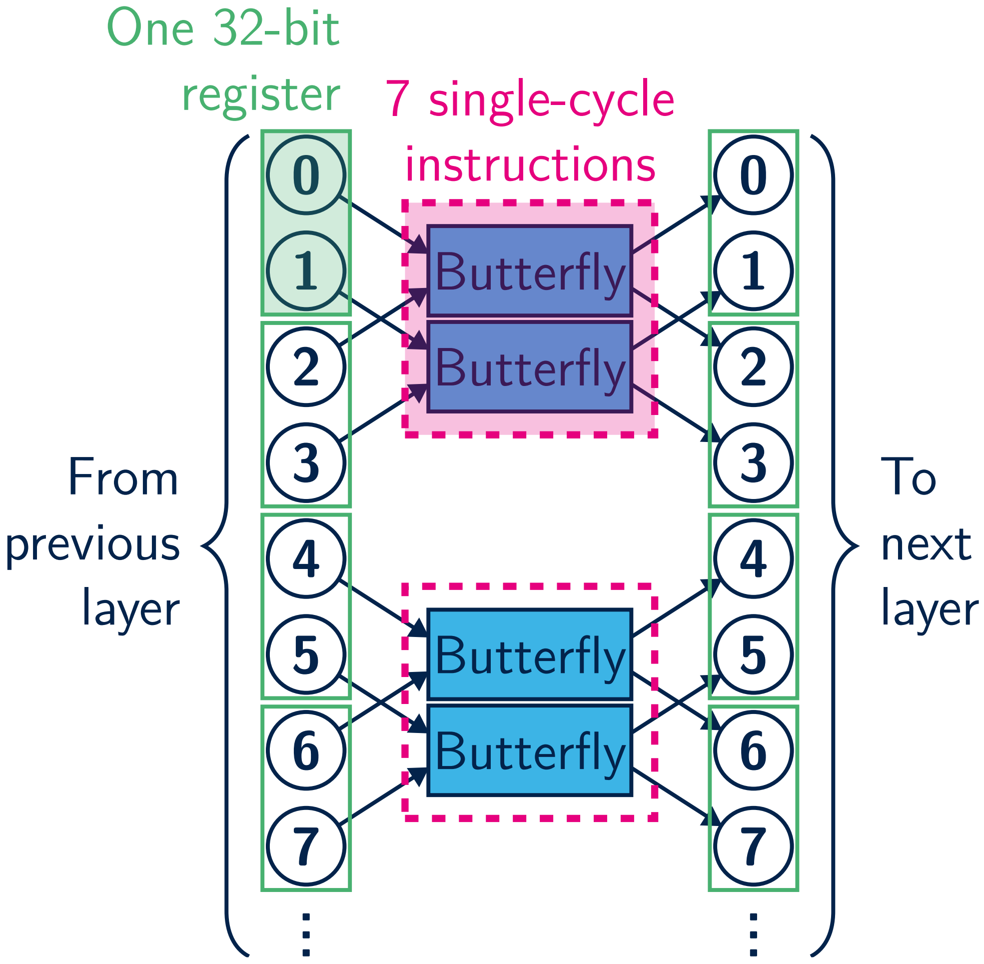
Key-recovery attack through **Inverse** NTT

Optimized NTT implementation for Cortex-M4

- Cortex-M4 microcontrollers have **DSP instructions** operating on **packed half-words**

- They can efficiently implement Kyber NTT **two butterflies at a time** by packing pairs of 16-bit coefficients into a 32-bit word [4]
  Kyber modulus has 12 bits, but coefficients can grow to 16 bits due to lazy reduction

- Previous BP attacks against the NTT did not use the same instructions and were not adapted to packed coefficients



[4] Huang *et al*., "Improved Plantard arithmetic for lattice-based cryptography," TCHES 2022/4.

Assembly implementation of the double butterfly

Double Cooley-Tukey butterfly for Cortex-M4

Input: Packed pairs of signed 16-bit coefficients $a = a_\mathsf{t} \parallel a_\mathsf{b}$, $b = b_\mathsf{t} \parallel b_\mathsf{b}$

Input: 32-bit corrected twiddle factor ζ (from real twiddle factor ζ_0)

Input: q and $q2^\alpha$ in two separate registers

Output: $a \equiv (a_\mathsf{t} + b_\mathsf{t}\zeta_0) \parallel (a_\mathsf{b} + b_\mathsf{b}\zeta_0)$, $b \equiv (a_\mathsf{t} - b_\mathsf{t}\zeta_0) \parallel (a_\mathsf{b} - b_\mathsf{b}\zeta_0)$

smulwb t, ζ, b — $// \ t = \zeta b_\mathsf{b} \gg 16$

smulwt b, ζ, b — $// \ b = \zeta b_\mathsf{t} \gg 16$

smlabb $t, t, q, q2^\alpha$ — $// \ t = t_\mathsf{b} q + q2^\alpha$

smlabb $b, b, q, q2^\alpha$ — $// \ b = b_\mathsf{b} q + q2^\alpha$

pkhtb $t, b, t, \mathrm{asr}\#16$ — $// \ t = t_\mathsf{b} \parallel b_\mathsf{b}$

usub16 b, a, t — $// \ b = (a_\mathsf{t} - t_\mathsf{t}) \parallel (a_\mathsf{b} - t_\mathsf{b})$

uadd16 a, a, t — $// \ a = (a_\mathsf{t} + t_\mathsf{t}) \parallel (a_\mathsf{b} + t_\mathsf{b})$

return a, b

The results written back by instructions are measured and accounted for in factors of types L (2 instances), I and V (2 instances).

The optimized Cortex-M4 implementation of [4] uses Plantard modular reduction.

Lazy reduction is used:

- the **results of modular multiplications** *are* ***reduced*** during the NTT
- the **results of modular additions/subtractions** *are* ***not reduced*** during the NTT: they are allowed to grow up to 16 bits and are only reduced during the subsequent point-wise multiplication.

[4] Huang *et al.*, “Improved Plantard arithmetic for lattice-based cryptography,” TCHES 2022/4.

Details on the attack environment

- **Software environment**
  - All attack phases are run by Python 3.10.6
  - The computation of messages is JIT-compiled using numba 0.55.1

- **Machine and resources**
  - AMD EPYC 7713P CPU @ 2 GHz, using 32 out of 64 cores (the other cores are used by unrelated tasks)
  - RAM usage peaked at 10 GB
  - All reported times are active CPU time, i.e. in core×hours. Real time is less due to parallelism.

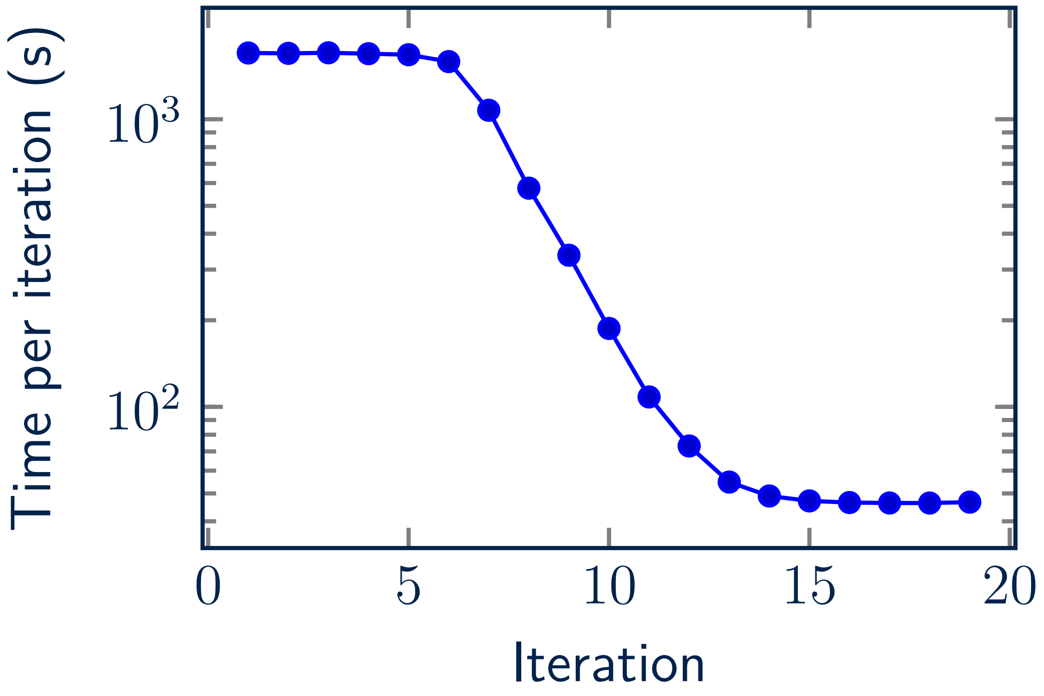
Effects of pruning on runtime

After each message update, we clear all outcomes having probability less than 10^{-8} times the probability of the most-likely outcome

→ Unlikely outcomes are progressively eliminated, making later iterations faster

Due to damping, little to no pruning takes place during the first iterations

Runtime per iteration for a typical execution of the attack against NTT with binomially distributed input, noise $\sigma = 5$



Details on damping and pruning, and stop conditions

- **Damping**: the new value of a message is computed as $m_{i+1} = (1 - \delta)m_i + \delta m'$ where $\delta = 0.95$, m_i is the old value of the message, and m' is given by the message-update equation

- **Pruning**: after each message update, we clear all outcomes having probability less than 10^{-8} times the probability of the most-likely outcome

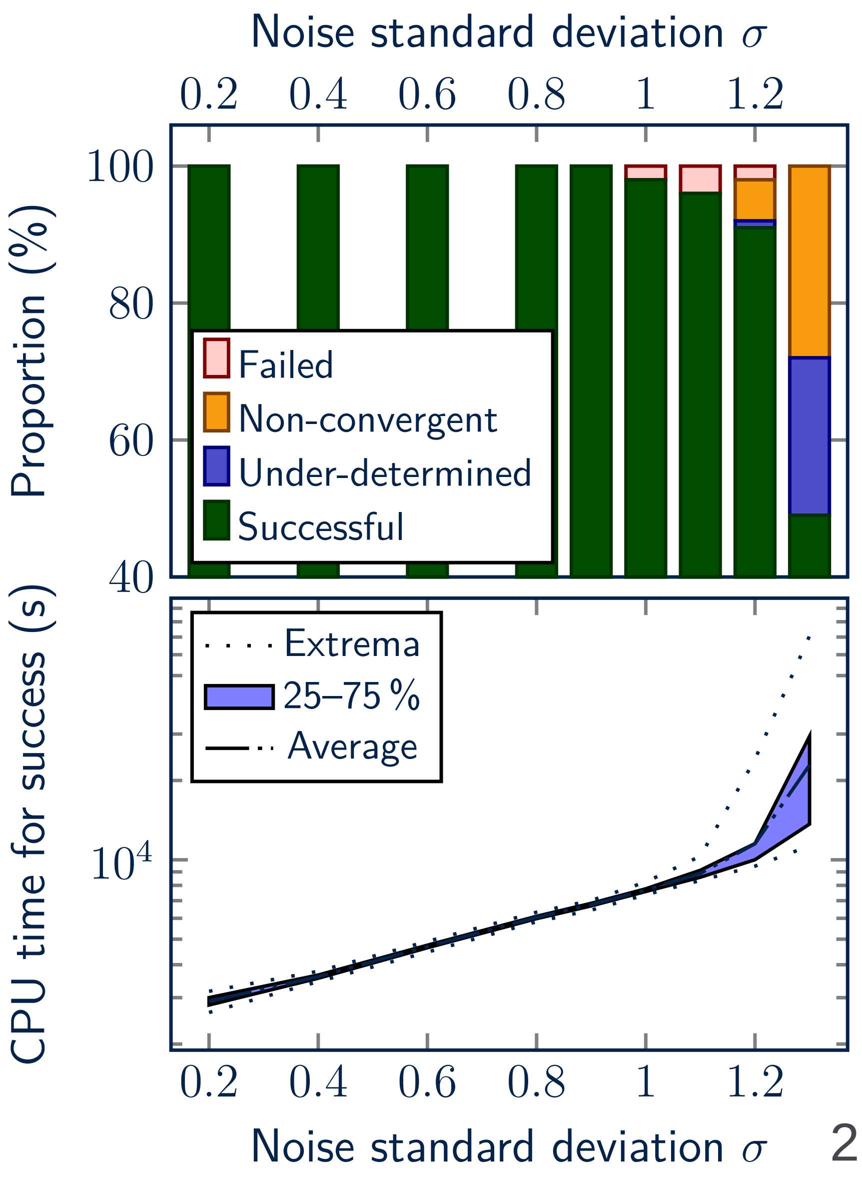
- We stop when reaching any of the **stop conditions**:
  - All messages changed by less than 10^{-5} (absolute value) during the previous iteration, or
  - A message having all-zero outcomes is computed, or
  - 100 iterations have been performed

Possible ending conditions

Four main ending conditions are observed

- **Successful**: belief propagation converges to a single possibility, and the highest-probability outcome is right for all variables. In most such cases, less than 20 iterations have been performed
- **Under-determined**: belief propagation reaches a stable state (messages no longer change) but many outcomes have nonzero probability
- **Non-convergent**: the iteration limit is reached but the messages still change across iterations
- **Failed**: an all-zero message is computed

Situation for a uniform-input NTT→



life.augmented