Persistent Fault Analysis

The Persistent Threat

Shivam Bhasin

Temasek Labs
NTU, Singapore

FDTC 2024

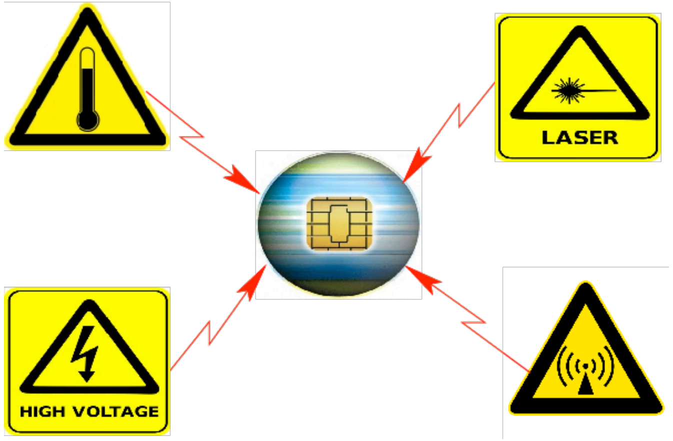
Table of Contents

1. Introduction to Fault Attacks
2. Persistent Fault Analysis (PFA)
3. PFA on Higher-Order Masking
4. Fault Attack on Lattice based PQC
5. Conclusions

Table of Contents

1. Introduction to Fault Attacks
2. Persistent Fault Analysis (PFA)
3. PFA on Higher-Order Masking
4. Fault Attack on Lattice based PQC
5. Conclusions

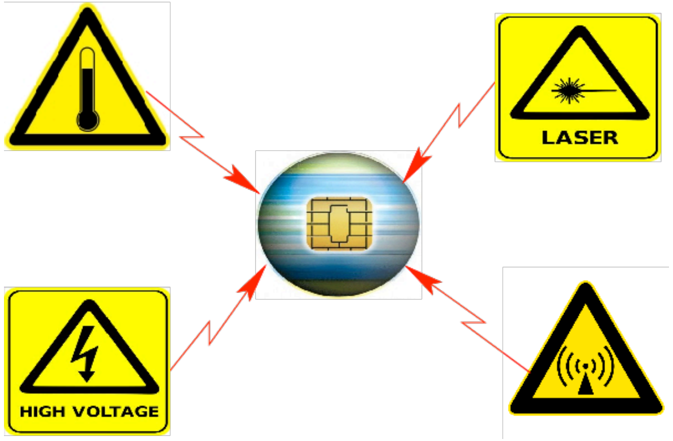
Fault Injection Attacks (FIA)



What is FIA?

- Physical Attacks
- Actively disturbs functioning of the target
- Exploits erroneous behavior

Fault Injection Attacks (FIA)



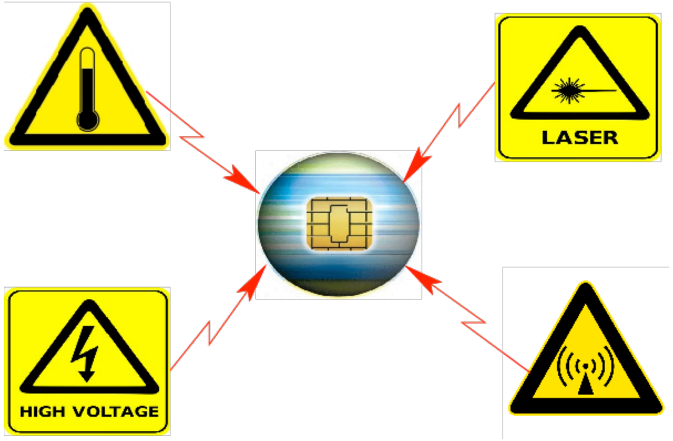
Injection Methods

- Global/Low-Cost/Low-Precision
  - Clock/Voltage glitch, temperature
- Local/High-Cost/High-Precision
  - Laser, Electromagnetic, Ion Beam

What is FIA?

- Physical Attacks
- Actively disturbs functioning of the target
- Exploits erroneous behavior

Fault Injection Attacks (FIA)



What is FIA?

- Physical Attacks
- Actively disturbs functioning of the target
- Exploits erroneous behavior

Injection Methods

- Global/Low-Cost/Low-Precision
  - Clock/Voltage glitch, temperature
- Local/High-Cost/High-Precision
  - Laser, Electromagnetic, Ion Beam

Impacts

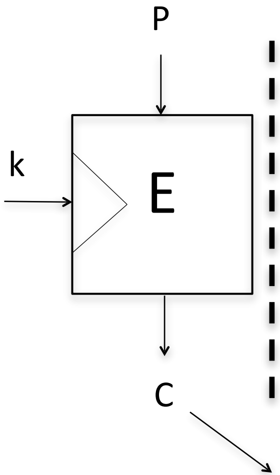
- Duration
  - Transient or Harmonic
- Effects
  - Data or Flow Modification
- Objectives
  - Corrupt computation, bypass security checks

Fault Analysis

- Differential Fault Analysis (DFA)
- Statistical Fault Analysis (SFA)

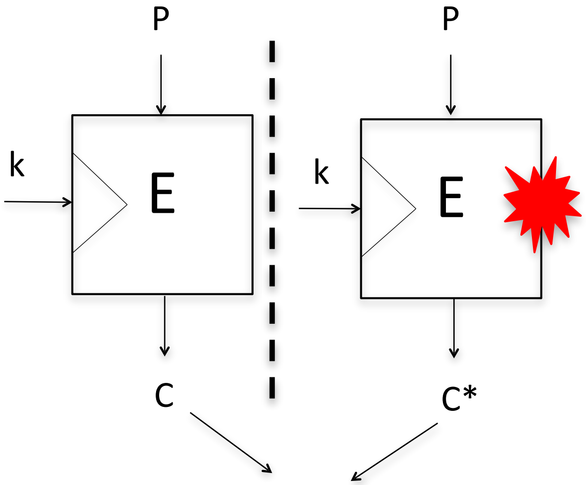
Fault Analysis

- Differential Fault Analysis (DFA)
- Statistical Fault Analysis (SFA)



Fault Analysis

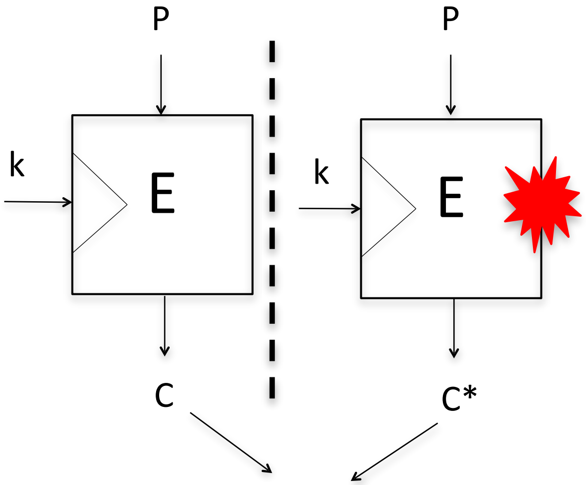
- Differential Fault Analysis (DFA)
- Statistical Fault Analysis (SFA)



Fault Analysis

- Differential Fault Analysis (DFA)
- Usually few ciphertext pair
- Control over plaintext needed

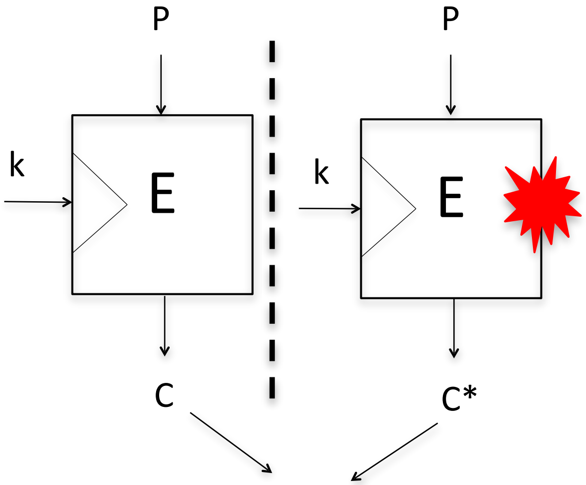
- Statistical Fault Analysis (SFA)



Analysis K=f(C,C*)

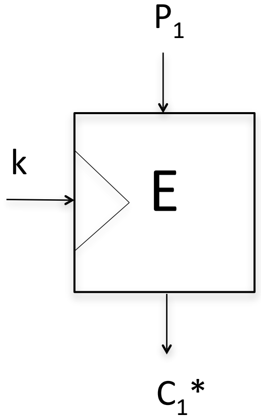
Fault Analysis

- Differential Fault Analysis (DFA)
- Usually few ciphertext pair
- Control over plaintext needed



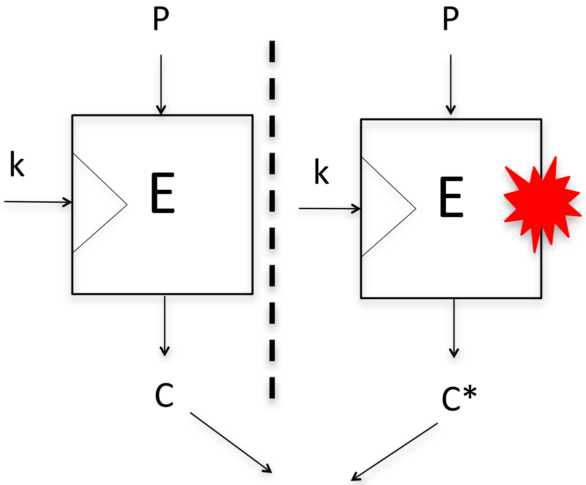
Analysis K=f(C,C*)

- Statistical Fault Analysis (SFA)



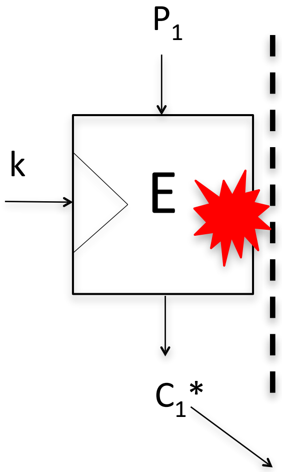
Fault Analysis

- Differential Fault Analysis (DFA)
- Usually few ciphertext pair
- Control over plaintext needed



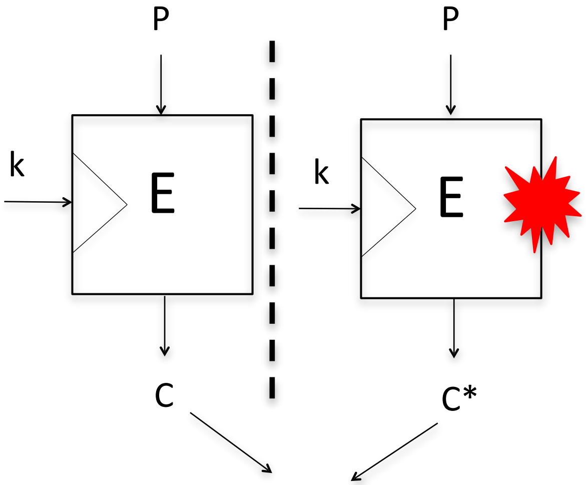
Analysis K=f(C,C*)

- Statistical Fault Analysis (SFA)



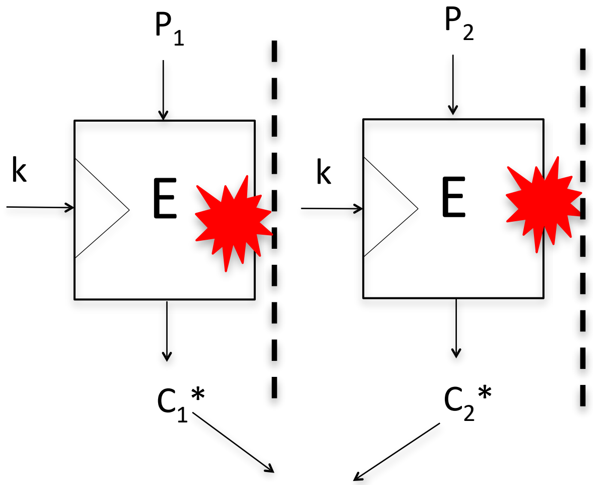
Fault Analysis

- Differential Fault Analysis (DFA)
- Usually few ciphertext pair
- Control over plaintext needed



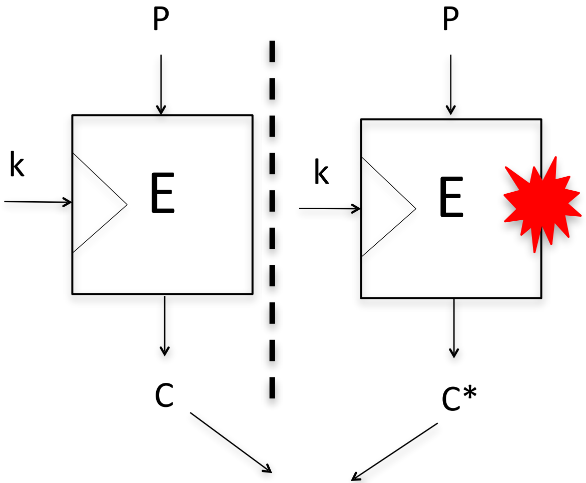
Analysis K=f(C,C*)

- Statistical Fault Analysis (SFA)



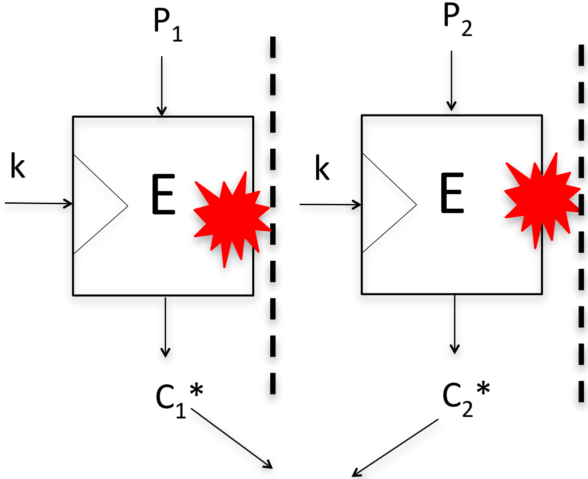
Fault Analysis

- Differential Fault Analysis (DFA)
- Usually few ciphertext pair
- Control over plaintext needed



Analysis K=f(C,C*)

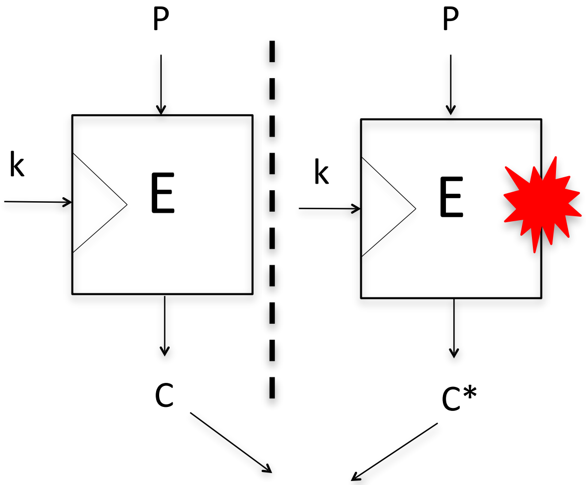
- Statistical Fault Analysis (SFA)



Analysis K=f(C_1*,C_2*, ...)

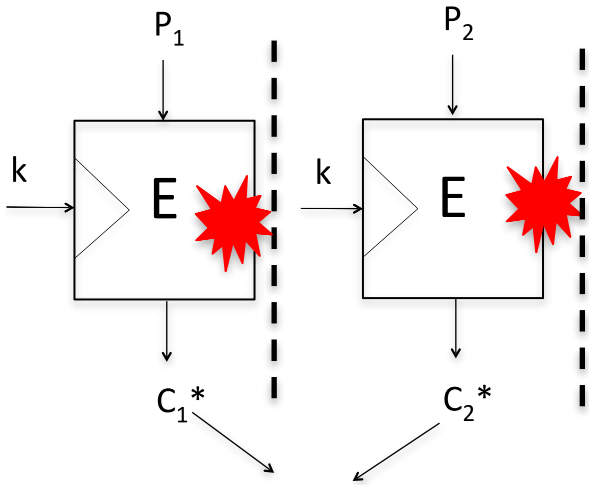
Fault Analysis

- Differential Fault Analysis (DFA)
- Usually few ciphertext pair
- Control over plaintext needed



Analysis K=f(C,C*)

- Statistical Fault Analysis (SFA)
- Need several ciphertext
- Several variants exist



Analysis $K=f(C_1^*,C_2^*, \dots)$

Limitations of SoA

- Very tight time synchronization on the round calculation and the injection timing

Limitations of SoA

- Very tight time synchronization on the round calculation and the injection timing
- Very complicated analysis due to the random value and the fault propagation

Limitations of SoA

- Very tight time synchronization on the round calculation and the injection timing
- Very complicated analysis due to the random value and the fault propagation
- May not work if there are countermeasures against fault attacks

Table of Contents

1. Introduction to Fault Attacks
2. Persistent Fault Analysis (PFA)
3. PFA on Higher-Order Masking
4. Fault Attack on Lattice based PQC
5. Conclusions

Revisiting Fault types

[1] Zhang, Fan, Xiaoxuan Lou, Xinjie Zhao, Shivam Bhasin, Wei He, Ruyi Ding, Samiya Qureshi, and Kui Ren. "Persistent fault analysis on block ciphers." IACR Transactions on Cryptographic Hardware and Embedded Systems (2018): 150-172.

Revisiting Fault types

- Transient: Affect one encryption
- Permanent: Always present

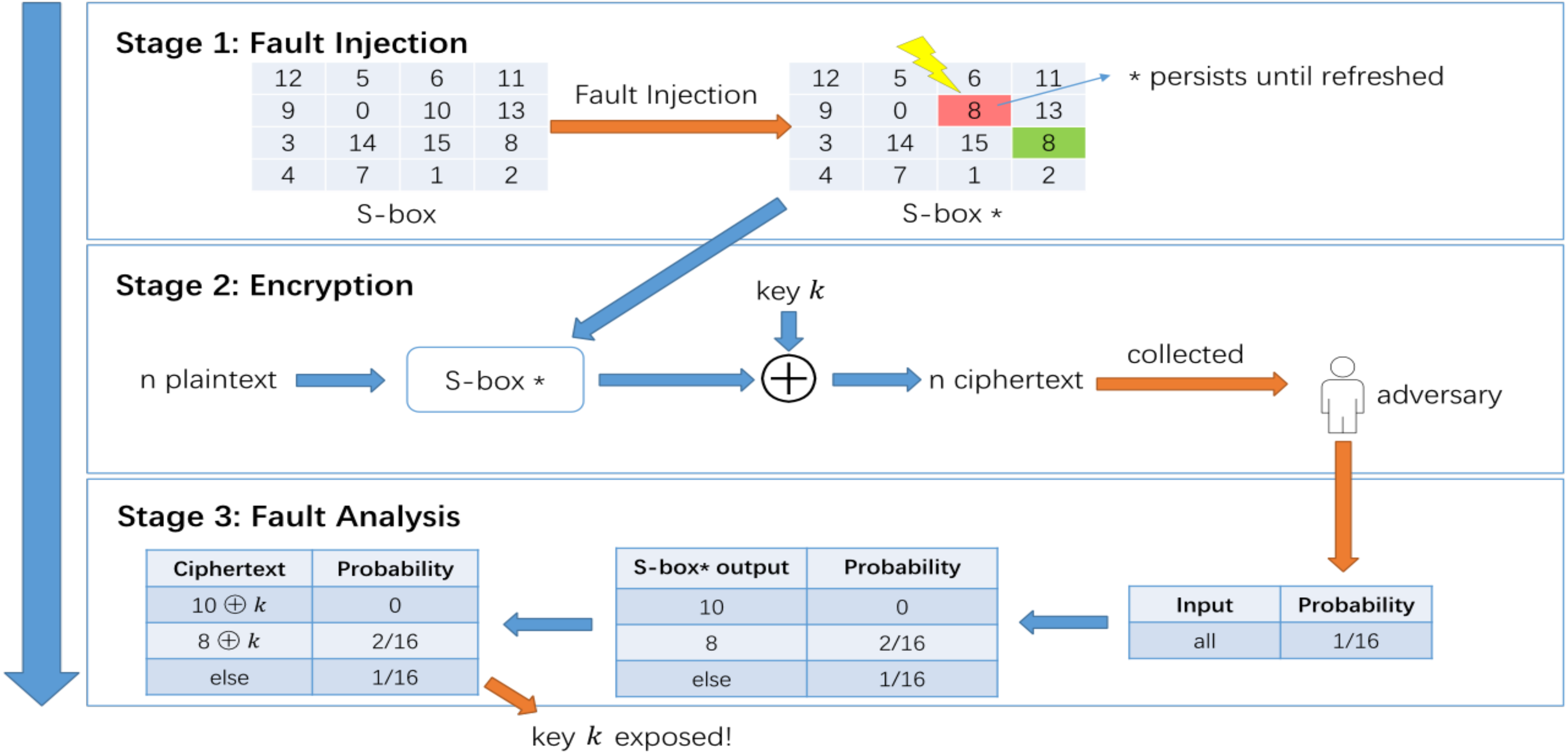
- **Persistent[1]:** Hybrid model between transient and permanent. Persist over several encryptions but disappears on reboot. Typically targets stored constants (ex. Sbox in memory)

[1] Zhang, Fan, Xiaoxuan Lou, Xinjie Zhao, Shivam Bhasin, Wei He, Ruyi Ding, Samiya Qureshi, and Kui Ren. "Persistent fault analysis on block ciphers." IACR Transactions on Cryptographic Hardware and Embedded Systems (2018): 150-172.

Adversary Model

- Block cipher with serial implementation
- Common Sbox as look-up table
- Persistent fault injected in one Sbox element
- Victim encrypts n plaintext with faulty Sbox
- Adversary can observe the n ciphertext
- No control on plaintext, except varying plaintext

Persistent Fault Analysis: Main Idea



Stage 1: Fault Injection

12	5	6	11
9	0	10	13
3	14	15	8
4	7	1	2

S-box

Fault Injection

12	5	6	11
9	0	8	13
3	14	15	8
4	7	1	2

S-box *

* persists until refreshed

Stage 2: Encryption

Stage 3: Fault Analysis

Input	Probability
all	1/16

S-box* output	Probability
10	0
8	2/16
else	1/16

Ciphertext	Probability
$10 \oplus k$	0
$8 \oplus k$	2/16
else	1/16

key k exposed!

PFA: Modus Operandi

- Statistical analysis on last round with ciphertext only

PFA: Modus Operandi

- Statistical analysis on last round with ciphertext only
- Fault changes one element $x \rightarrow x^*$ in Sbox (lets say 4X4 Sbox)

PFA: Modus Operandi

- Statistical analysis on last round with ciphertext only
- Fault changes one element x→x* in Sbox (lets say 4X4 Sbox)
- Expectation $E(x)= 0$, $E(x^*)=2/16$, $E(y\neq(x,x^*))=1/16$

PFA: Modus Operandi

- Statistical analysis on last round with ciphertext only
- Fault changes one element x→x* in Sbox (lets say 4X4 Sbox)
- Expectation $E(x)= 0$, $E(x^*)=2/16$, $E(y \neq (x,x^*))=1/16$
- Three analysis strategies:

PFA: Modus Operandi

- Statistical analysis on last round with ciphertext only
- Fault changes one element x→x* in Sbox (lets say 4X4 Sbox)
- Expectation $E(x)= 0$, $E(x^*)=2/16$, $E(y\neq(x,x^*))=1/16$
- Three analysis strategies:
  - t_{min}: find the missing value in Sbox table (x). Then $k = t_{min} \oplus x$;

PFA: Modus Operandi

- Statistical analysis on last round with ciphertext only
- Fault changes one element x→x* in Sbox (lets say 4X4 Sbox)
- Expectation $E(x) = 0$, $E(x^*) = 2/16$, $E(y \neq (x,x^*)) = 1/16$
- Three analysis strategies:
  - t_{min}: find the missing value in Sbox table (x). Then $k = t_{min} \oplus x$;
  - $t \neq t_{min}$: find values t where $t \neq t_{min}$ and eliminate candidates for k;

PFA: Modus Operandi

- Statistical analysis on last round with ciphertext only
- Fault changes one element $x \rightarrow x^*$ in Sbox (lets say 4X4 Sbox)
- Expectation $E(x)= 0$, $E(x^*)=2/16$, $E(y \neq (x,x^*))=1/16$
- Three analysis strategies:
  - t_{min}: find the missing value in Sbox table (x). Then $k = t_{min} \oplus x$;
  - $t \neq t_{min}$: find values t where $t \neq t_{min}$ and eliminate candidates for k;
  - t_{max}: find the value with max probability (x'). Then $k = t_{max} \oplus x^*$

PFA: Modus Operandi

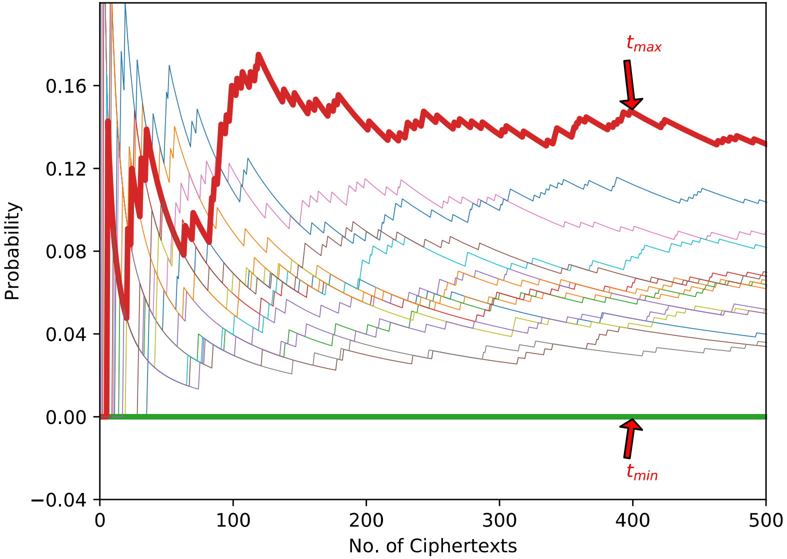
- Statistical analysis on last round with ciphertext only
- Fault changes one element $x \rightarrow x^*$ in Sbox (lets say 4X4 Sbox)
- Expectation $E(x)= 0$, $E(x^*)=2/16$, $E(y \neq (x,x^*))=1/16$
- Three analysis strategies:
  - t_{min}: find the missing value in Sbox table (x). Then $k = t_{min} \oplus x$;
  - $t \neq t_{min}$: find values t where $t \neq t_{min}$ and eliminate candidates for k;
  - t_{max}: find the value with max probability (x’). Then $k = t_{max} \oplus x^*$
- No. of ciphertext n can be determined by coupon collector’s problem

PFA: Modus Operandi

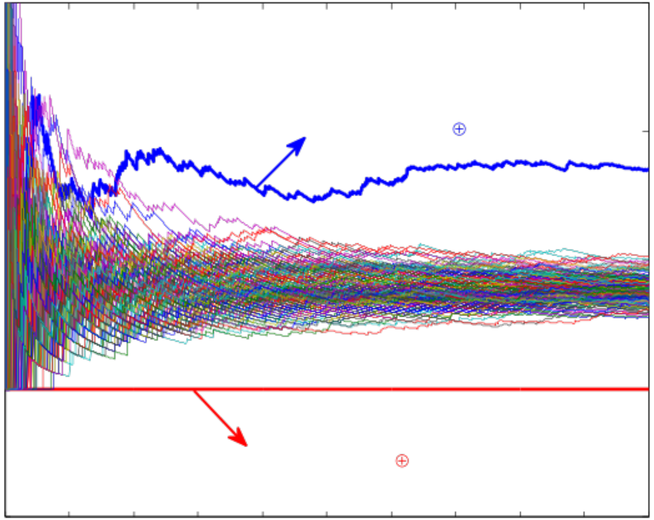
- Statistical analysis on last round with ciphertext only
- Fault changes one element x→x* in Sbox (lets say 4X4 Sbox)
- Expectation $E(x)= 0$, $E(x^*)=2/16$, $E(y \neq (x,x^*))=1/16$
- Three analysis strategies:
  - t_{min}: find the missing value in Sbox table (x). Then $k = t_{min} \oplus x$;
  - $t \neq t_{min}$: find values t where $t \neq t_{min}$ and eliminate candidates for k;
  - t_{max}: find the value with max probability (x'). Then $k = t_{max} \oplus x^*$
- No. of ciphertext n can be determined by coupon collector's problem
- x, x* can be brute-forced if not known

PFA on PRESENT and AES

PRESENT: n ≥50

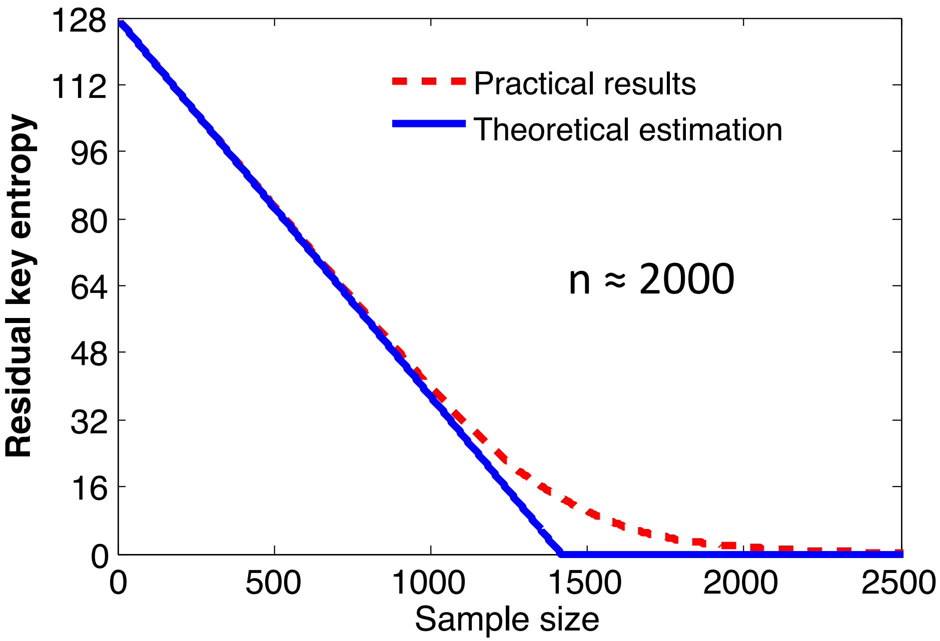


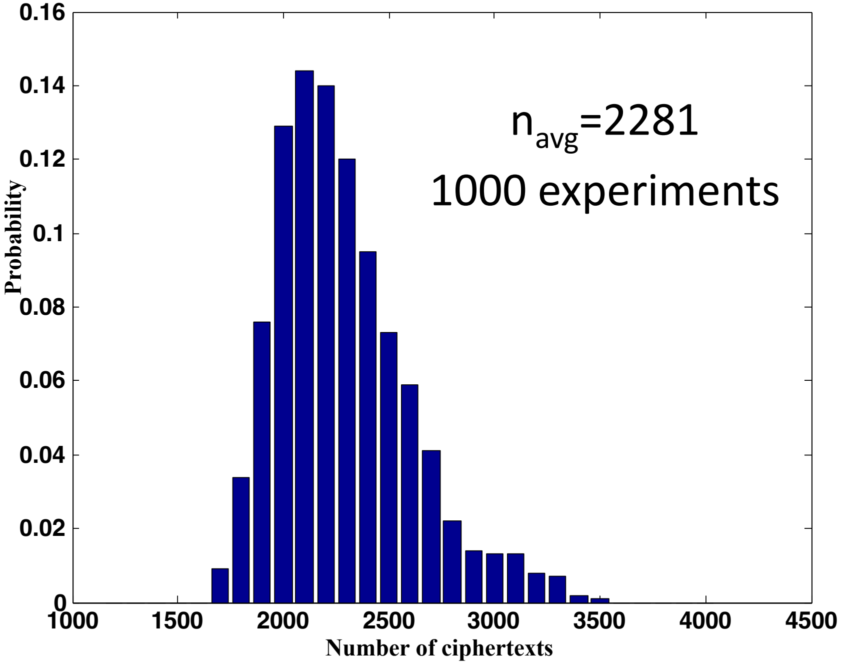
AES: n ≥ 1560



n= Minimum no of ciphertext needed by coupon collector's problem

Practical PFA on AES





n_{avg}=2281

1000 experiments

Comparison vs Other Fault Attacks

➕

(1) The attack is not differential in nature and thus the control over the plaintext is not required.

(2) The adversary does not necessarily need live synchronization

(3) The fault model remains relaxed (no biased faults needed)

(4) PFA can also be applied in multiple fault setting

(5) PFA can bypass some redundancy based countermeasures

(6) An adversary can always inject the persistent fault before the victim is switched to the sensitive mode

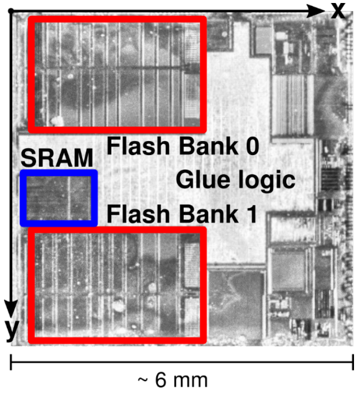
➖

(1) It needs higher number of ciphertexts as compared to DFA

(2) Persistent faults can be detected by some built-in health test mechanism or fault counters.

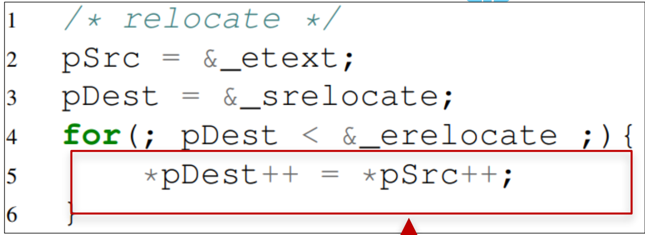
T-table corruption

- EM fault on ARM Cortex-M3 with 100% repeatability
- Public AES implementation from Schwabe and Stoffelen
- Single T-table,
- 4 columns of 32 bits in the data buffer



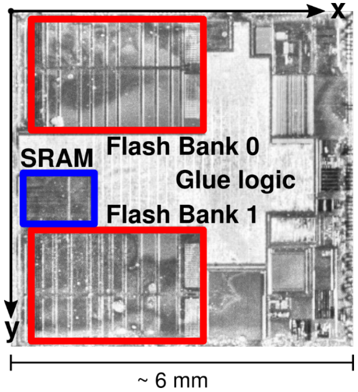
T-table corruption

- EM fault on ARM Cortex-M3 with 100% repeatability
- Public AES implementation from Schwabe and Stoffelen
- Single T-table,
- 4 columns of 32 bits in the data buffer

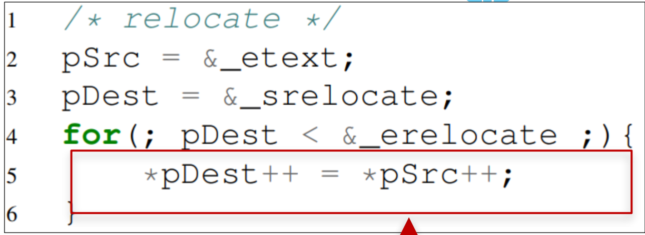


```
/* relocate */
pSrc = &_etext;
pDest = &_srelocate;
for(; pDest < &_erelocate ;){
    *pDest++ = *pSrc++;
}
```

128-bit Flash memory access



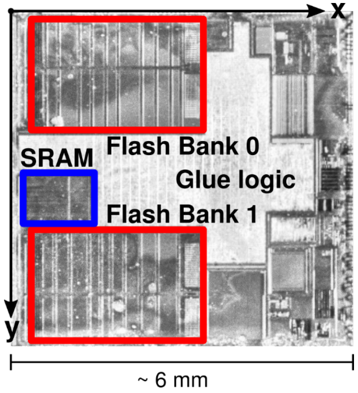
T-table corruption



```
/* relocate */
pSrc = &_etext;
pDest = &_srelocate;
for(; pDest < &_erelocate ;){
    *pDest++ = *pSrc++;
}
```

128-bit Flash memory access

- EM fault on ARM Cortex-M3 with 100% repeatability
- Public AES implementation from Schwabe and Stoffelen
- Single T-table,
- 4 columns of 32 bits in the data buffer

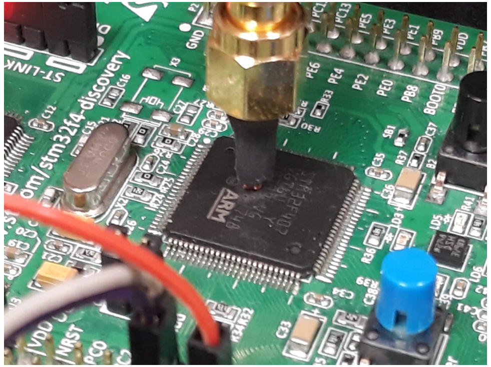


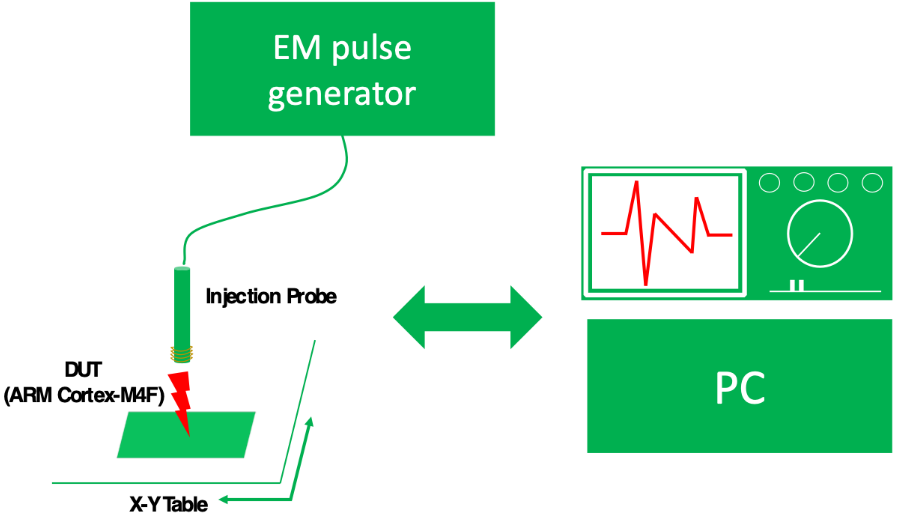
$$T[v] = \begin{bmatrix} S[v] \circ 01 \\ S[v] \circ 03 \\ S[v] \circ 02 \\ S[v] \circ 01 \end{bmatrix} \longrightarrow T[v^*] = \begin{bmatrix} a \\ a \circ 03 \\ a \circ 02 \\ a \end{bmatrix} \boxed{\Leftrightarrow a = 0}$$

Fault condition

Fault on 4 columns = residual key entropy of 32 bits (practical to brute-force)

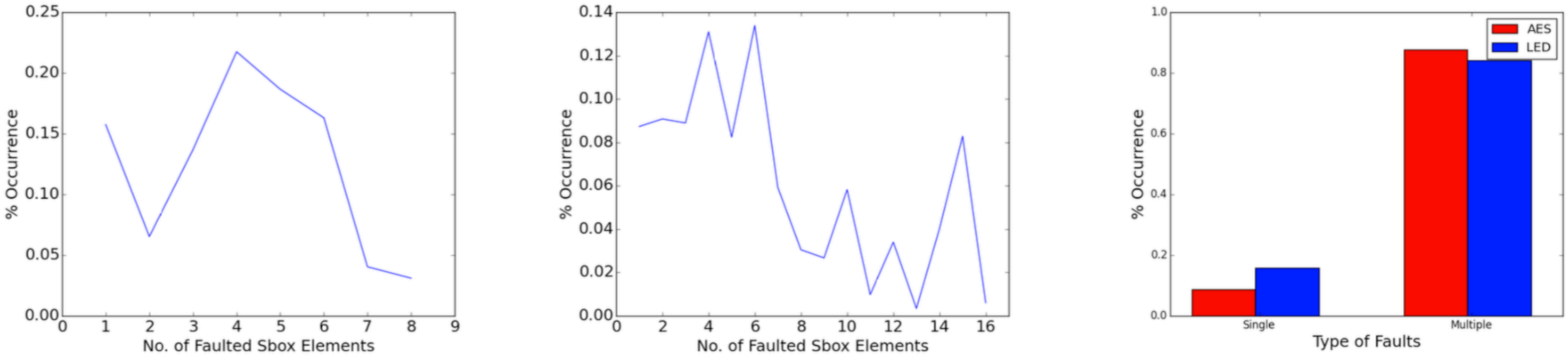
Experimental Setup





- Target Chip: Public implementation on STM32F407VG (opt –O3)
- Injection Time: Boot-up
- Target Operation: Sbox transfer from Flash to RAM

Experimental Results



(a) LED-64

(b) AES-128

(c) Single vs Multiple

Further Improvements

No. of Persistent Faults	Ciphertext Needed	Key Complexity	Reference
1	2273	2^0	Zhang et al. TCHES 2018
	1641	2^0	Zhang et al. TCHES 2020
	1000	2^0	Xu et al. TCAD 2021
2/16	7775/1643	2^{50} /2^{50}	Engels et al. FDTC 2020
	1552/477	2^{23} /$2^{24.5}$	Soleimany et al. TCHES 2022
	785/256	2^{16} /2^0	Zhang et al. TCHES 2023

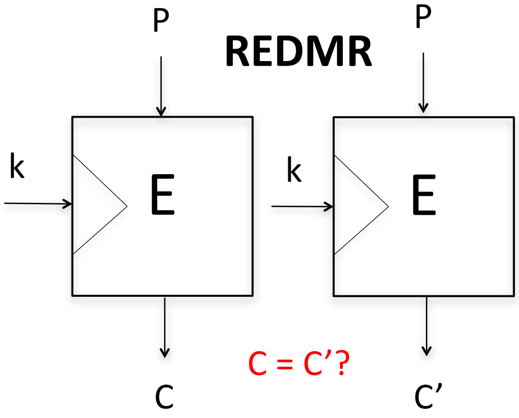
Other works further relax attack models, cipher construction

Dual Modular Redundancy (DMR) Countermeasure

- Compute twice and compare (REDMR)
- Compute forward-inverse and compare (IDDMR)
- If ≠
    - NCO: No Ciphertext output
    - ZVO: Zero Value output
    - RCO: Random Ciphertext output
- Provably secure against single fault
- Adversary can either target the encryption or comparison but not both
- REDMR broken by design if same S-box is used
- Lets target IDDMR, more difficult of the two

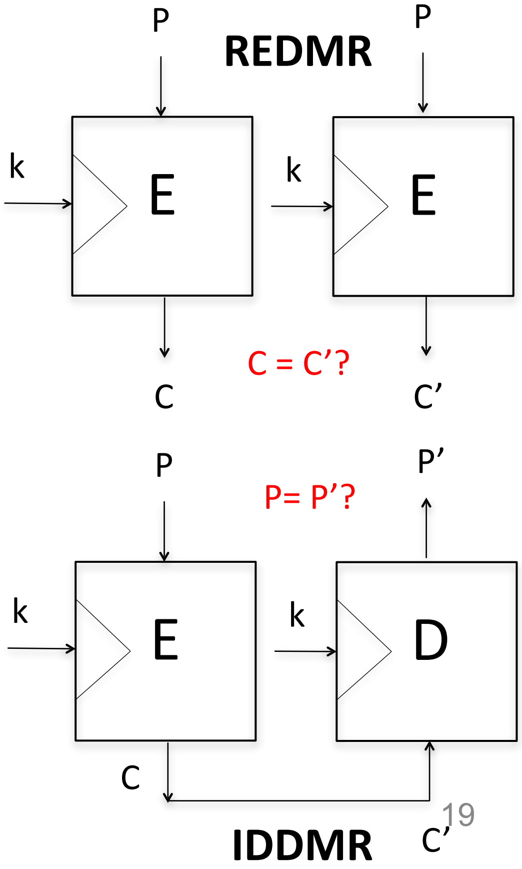
Dual Modular Redundancy (DMR) Countermeasure

- Compute twice and compare (REDMR)
- Compute forward-inverse and compare (IDDMR)
- If ≠
  - NCO: No Ciphertext output
  - ZVO: Zero Value output
  - RCO: Random Ciphertext output
- Provably secure against single fault
- Adversary can either target the encryption or comparison but not both
- REDMR broken by design if same S-box is used
- Lets target IDDMR, more difficult of the two



Dual Modular Redundancy (DMR) Countermeasure

- Compute twice and compare (REDMR)
- Compute forward-inverse and compare (IDDMR)
- If ≠
  - NCO: No Ciphertext output
  - ZVO: Zero Value output
  - RCO: Random Ciphertext output
- Provably secure against single fault
- Adversary can either target the encryption or comparison but not both
- REDMR broken by design if same S-box is used
- Lets target IDDMR, more difficult of the two



Attacking IDDMR with NCO/ZVO

- Faulty outputs are suppressed
- Some output will be not affected by fault
- Probability p of correct output is f(x,k)
- *p* for AES

$$p = (1 - \frac{1}{256})^{160} = 0.5346$$

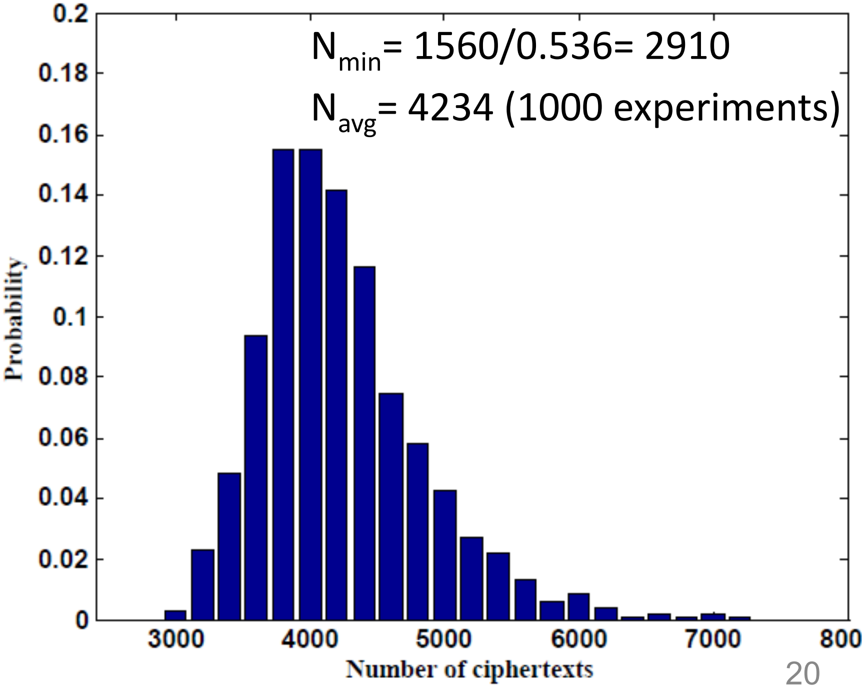
- Adversary roughly needs n/p ciphertext

Attacking IDDMR with NCO/ZVO

- Faulty outputs are suppressed
- Some output will be not affected by fault
- Probability p of correct output is f(x,k)
- *p* for AES

$$p = (1 - \frac{1}{256})^{160} = 0.5346$$

- Adversary roughly needs n/p ciphertext



Attacking IDDMR with RCO

- Faulty output is replaced by uniformly random
- Slight difference in distribution of random output and correct ciphertext
- The bias can be detected with more ciphertext (n)

$$Pr(y = x) = 0 \times p + \frac{1}{256} \times (1 - p) = \frac{0.4654}{256}$$

$$Pr(y = x^*) = \frac{2}{256} \times p + \frac{1}{256} \times (1 - p) = \frac{1.5346}{256}$$

$$Pr(y \neq x \wedge y \neq x^*) = \frac{1}{256} \times p + \frac{1}{256} \times (1 - p) = \frac{1}{256}$$

- Roughly n≈10000 resulted in attack success

Attacking IDDMR with RCO

- Faulty output is replaced by uniformly random
- Slight difference in distribution of random output and correct ciphertext
- The bias can be detected with more ciphertext (n)

$$Pr(y = x) = 0 \times p + \frac{1}{256} \times (1-p) = \frac{0.4654}{256}$$

$$Pr(y = x^*) = \frac{2}{256} \times p + \frac{1}{256} \times (1-p) = \frac{1.5346}{256}$$

$$Pr(y \neq x \wedge y \neq x^*) = \frac{1}{256} \times p + \frac{1}{256} \times (1-p) = \frac{1}{256}$$

- Roughly n≈10000 resulted in attack success

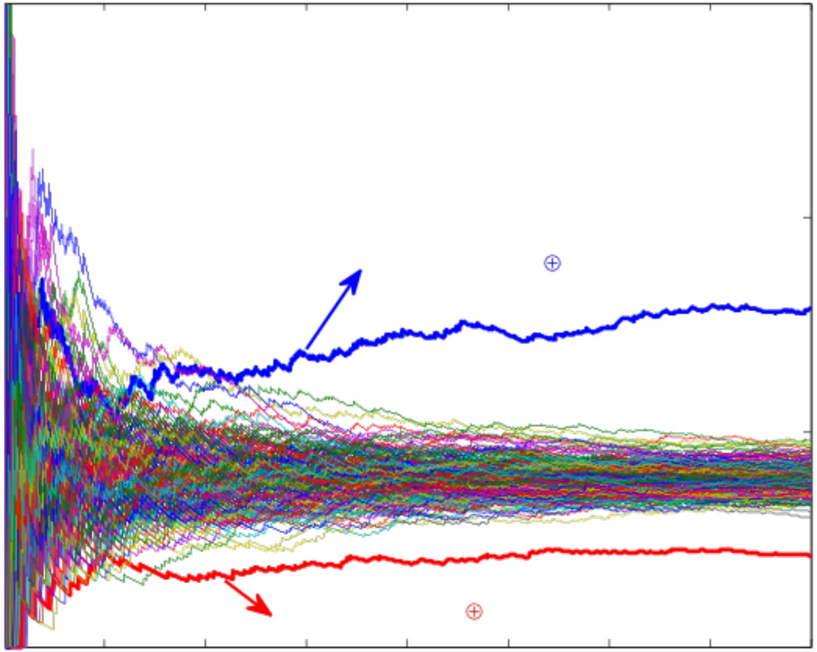


Table of Contents

1. Introduction to Fault Attacks
2. Persistent Fault Analysis (PFA)
3. PFA on Higher-Order Masking
4. Fault Attack on Lattice based PQC
5. Conclusions

Masking Countermeasure

- Masking is an algorithmic side-channel countermeasure
- Based on Shamir’s secret sharing
- Boolean Masking:
  - Secret x split into tuple (x_m, m)
  - $x_m = x \oplus m$
  - m is randomly chosen on each execution
  - For higher order masking m is split in further shares
  - At masking order d: $m = m_1 \oplus m_2 \oplus m_3 \ldots \oplus m_d$
- Removes dependency between x and side-channel leakage

Masking Countermeasure

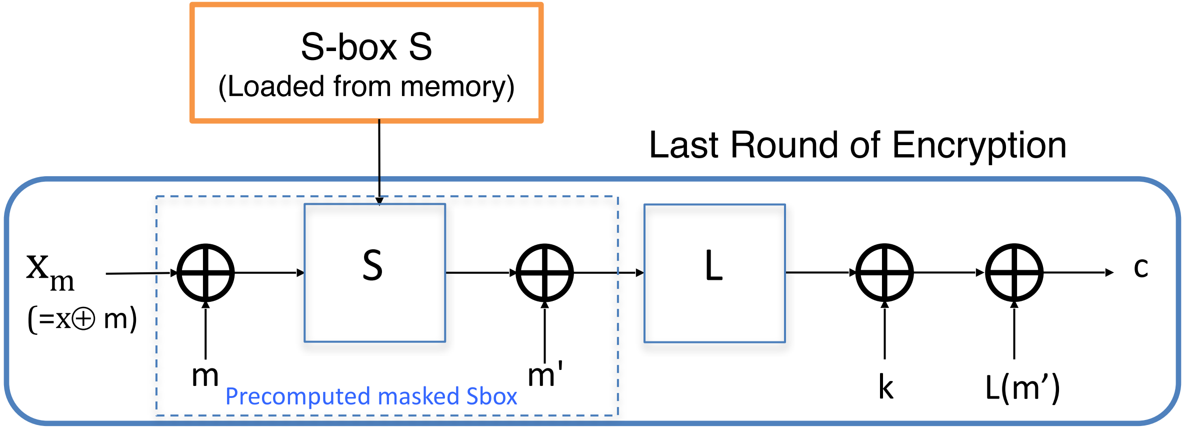
- Masking is an algorithmic side-channel countermeasure
- Based on Shamir's secret sharing
- Boolean Masking:
  - Secret x split into tuple (x_m,m)
  - $x_m = x \oplus m$
  - m is randomly chosen on each execution
  - For higher order masking m is split in further shares
  - At masking order d: $m = m_1 \oplus m_2 \oplus m_3 \ldots \oplus m_d$
- Removes dependency between x and side-channel leakage

[2] Pan, Jingyu, Fan Zhang, Kui Ren, and Shivam Bhasin. "One fault is all it needs: Breaking higher-order masking with persistent fault analysis." In 2019 Design, Automation & Test in Europe Conference & Exhibition (DATE), pp. 1-6. IEEE, 2019.

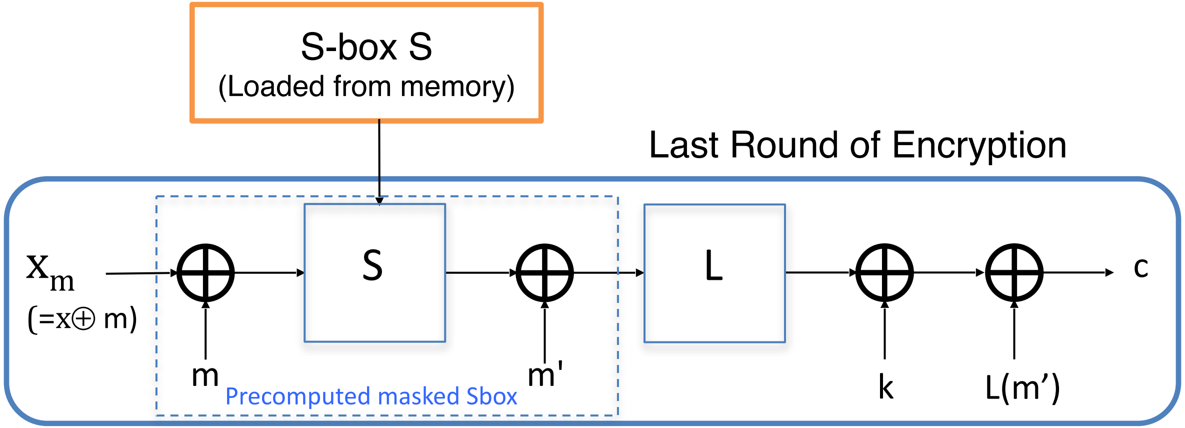
Masking vs PFA

- Theoretically masking does not resist fault attacks
- Several previous attack were presented on masking
- They work in restrictive setting (advanced fault model, high no. of faults etc.)
- **Only One Fault** to break 4 various **public implementation of masking**
- Target Implementations:
  - Byte-wise Masking [SES, Virginia Tech]
  - Coron’s Table Masking [EuroCrypt 2014]
  - Rivian and Prouff Masking [CHES 2010]
  - Software Threshold [COSADE 2018]

PFA on Masking: Generic Attack



PFA on Masking: Generic Attack



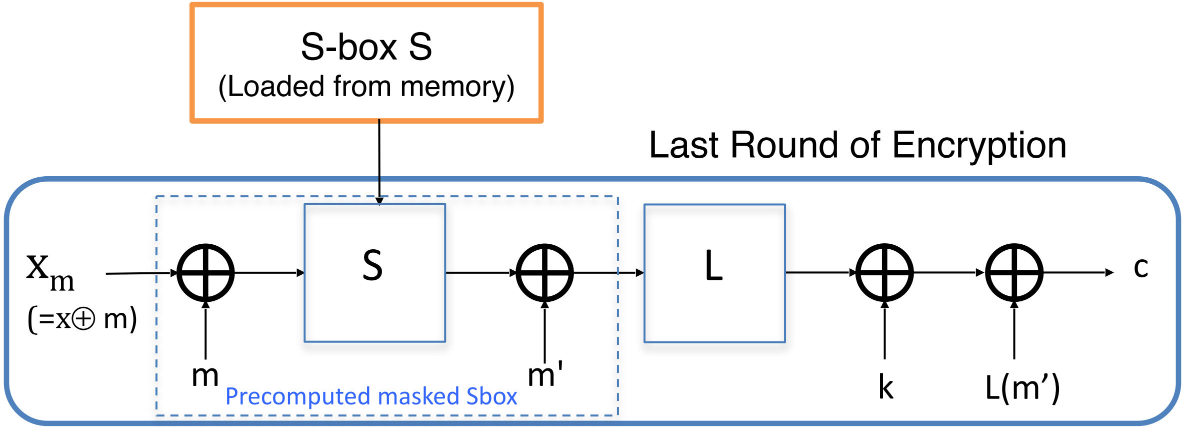
$$c = L(S(x_m \oplus m) \oplus m') \oplus k) \oplus L(m')$$

$$= L(S(x \oplus m \oplus m) \oplus m') \oplus k) \oplus L(m')$$

$$= L(S(x)) \oplus k \oplus L(m') \oplus L(m')$$

$$= L(S(x)) \oplus k$$

PFA on Masking: Generic Attack



$$c = L(S(x_m \oplus m) \oplus m') \oplus k) \oplus L(m')$$

$$= L(S(x \oplus m \oplus m) \oplus m') \oplus k) \oplus L(m')$$

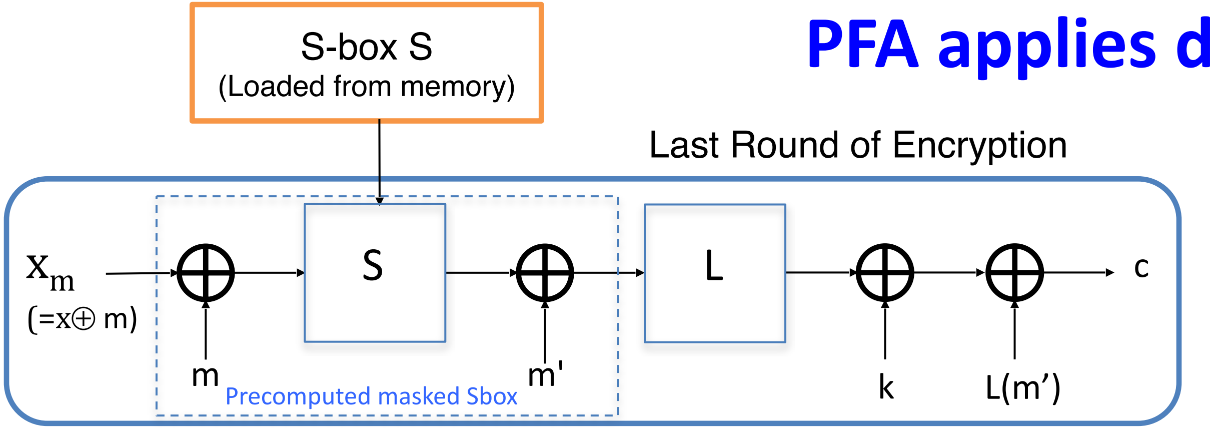
$$= L(S(x)) \oplus k \oplus L(m') \oplus L(m')$$

$$= L(S(x)) \oplus k$$

→

Masking has no effect on the distribution of the final ciphertext

PFA on Masking: Generic Attack

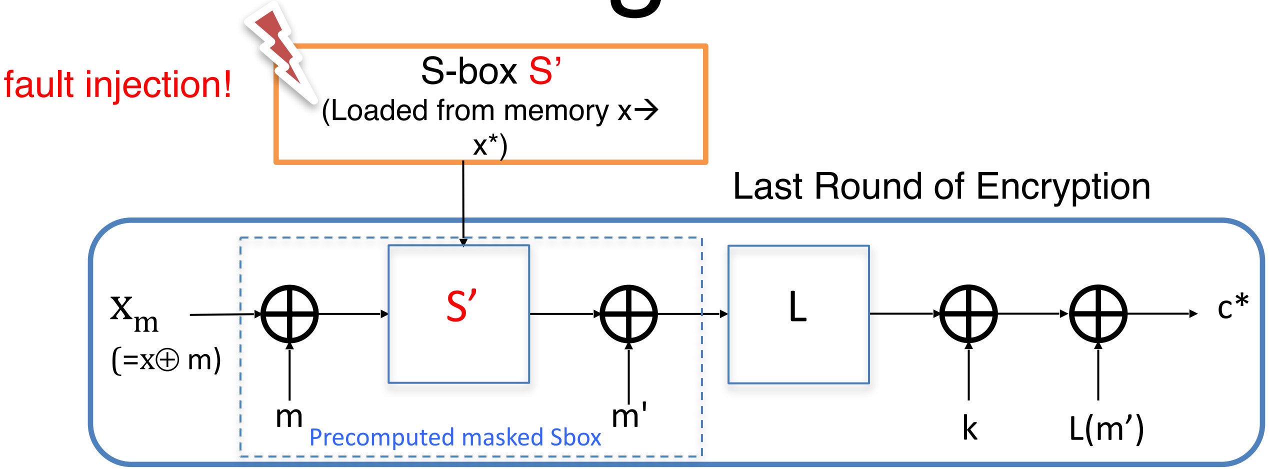


PFA applies directly!!!

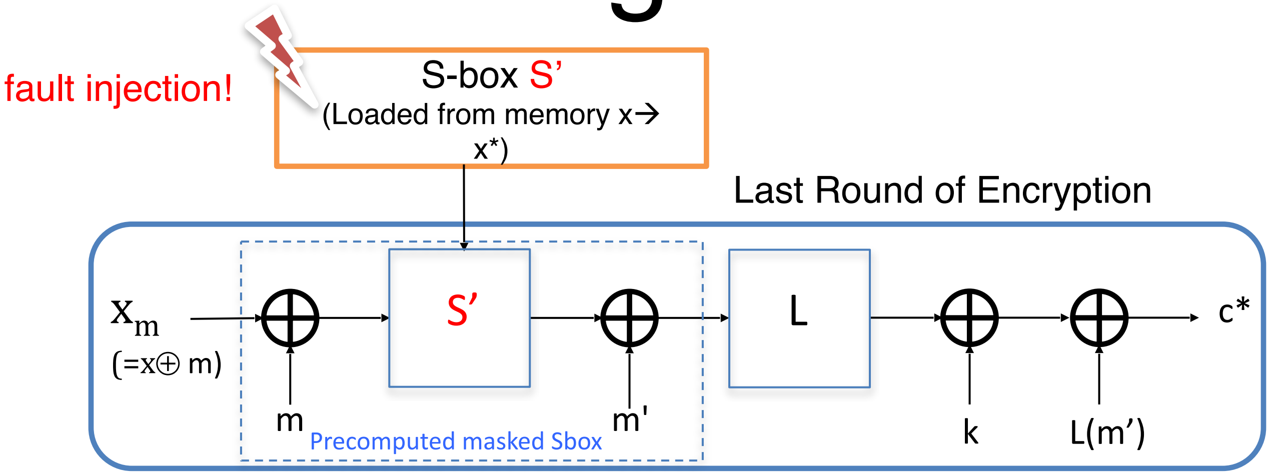
$$c = L(S(x_m \oplus m) \oplus m') \oplus k) \oplus L(m')$$
$$= L(S(x \oplus m \oplus m) \oplus m') \oplus k) \oplus L(m')$$
$$= L(S(x)) \oplus k \oplus L(m') \oplus L(m')$$
$$= L(S(x)) \oplus k$$

→ Masking has no effect on the distribution of the final ciphertext

PFA on Masking: Generic Attack



PFA on Masking: Generic Attack



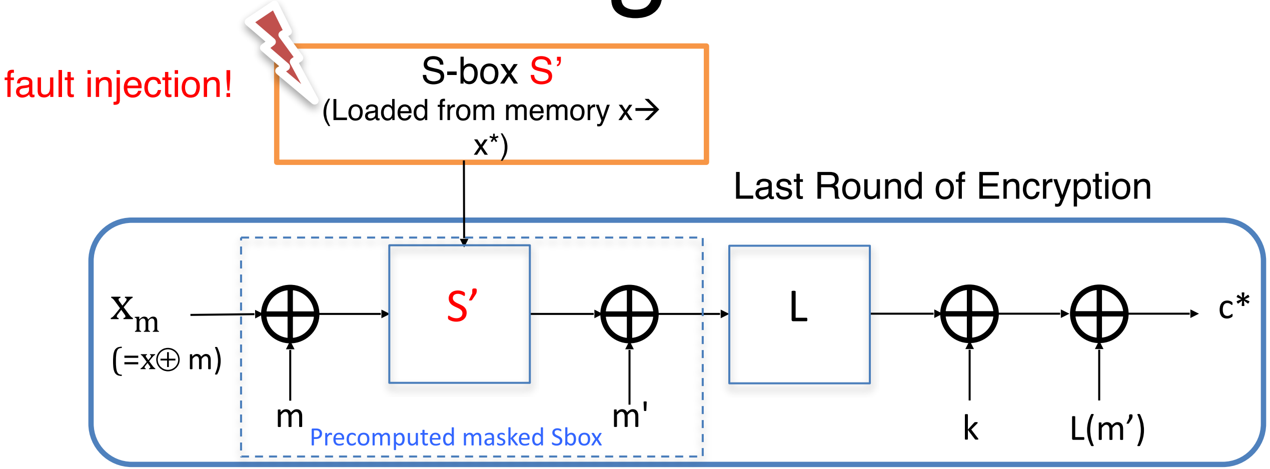
$$c^* = L(S'(x_m \oplus m) \oplus m') \oplus k) \oplus L(m')$$

$$= L(S'(x \oplus m \oplus m) \oplus m') \oplus k) \oplus L(m')$$

$$= L(S'(x)) \oplus k \oplus L(m') \oplus L(m')$$

$$= L(S'(x)) \oplus k$$

PFA on Masking: Generic Attack



$$
\begin{aligned}
c^* &= L(S'(x_m \oplus m) \oplus m') \oplus k) \oplus L(m') \\
&= L(S'(x \oplus m \oplus m) \oplus m') \oplus k) \oplus L(m') \\
&= L(S'(x)) \oplus k \oplus L(m') \oplus L(m') \\
&= L(S'(x)) \oplus k
\end{aligned}
$$

Value $c^*=L(S'(x) \oplus k)$ will be missing

Value $c^*=L(S'(x^*) \oplus k)$ will be doubled

→ Allows key recovery with PFA

m,m’ do not appear

Also masking order does not matter

Attack Results on Public Code

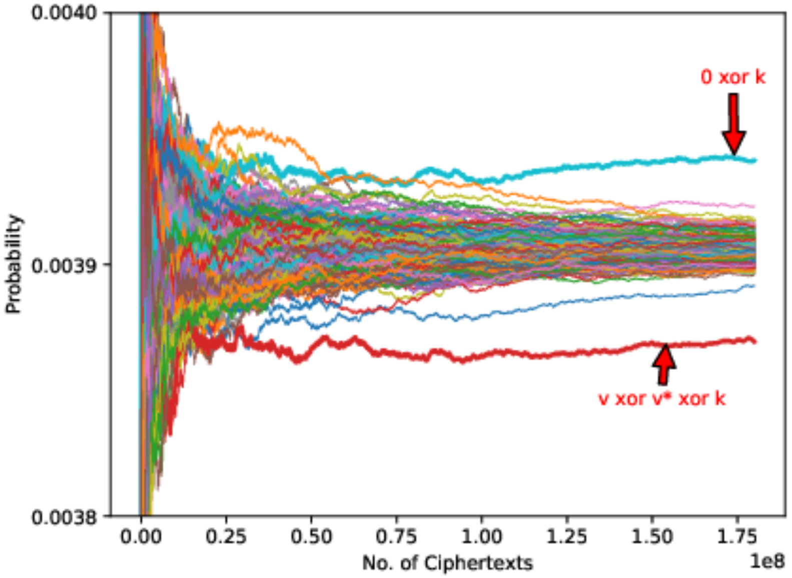
Design	Fault Target	No. of Ciphertext (Masking Order)
Bytewise Masking (Virginiatech)	Sbox Recomputation	1560 (any)
Coron's higher Order Masking (Eurocrypt 2014)	Sbox Recomputation	1560 (any)
Rivian & Prouff Masking (CHES 2010)	Affine transformation	2,500,000 (1) $[\alpha\ 2^{14d}]$
Software Threshold (COSADE 2018)	Decomposition A'''	400,000 (1)

Attack Results on Public Code

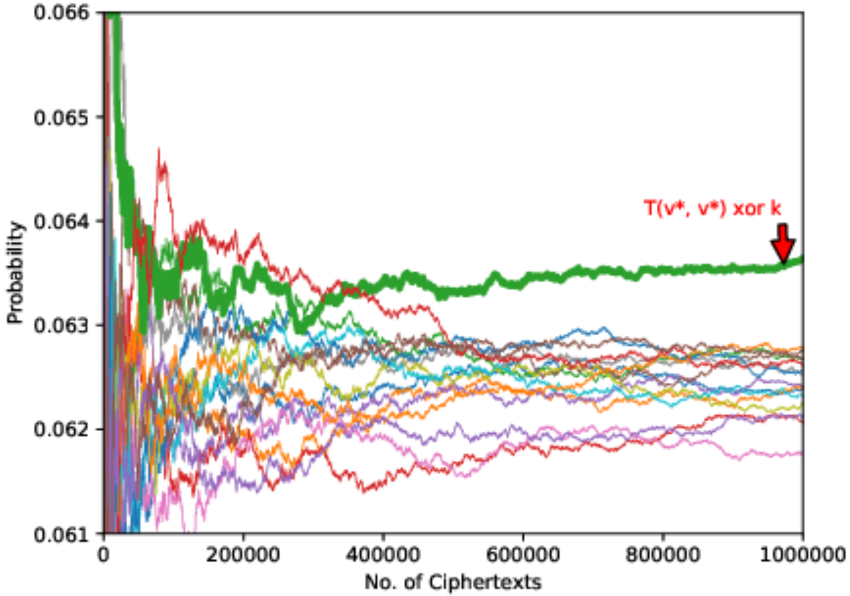
Design	Fault Target	No. of Ciphertext (Masking Order)
Bytewise Masking (Virginiatech)	Sbox Recomputation	1560 (any)
Coron’s higher Order Masking (Eurocrypt 2014)	Sbox Recomputation	1560 (any)
Rivian & Prouff Masking (CHES 2010)	Affine transformation	2,500,000 (1) [α 2^{14d}]
Software Threshold (COSADE 2018)	Decomposition A’’’	400,000 (1)

ONLY ONE FAULT

Attack Results on Public Code



Rivian & Prouff Masking

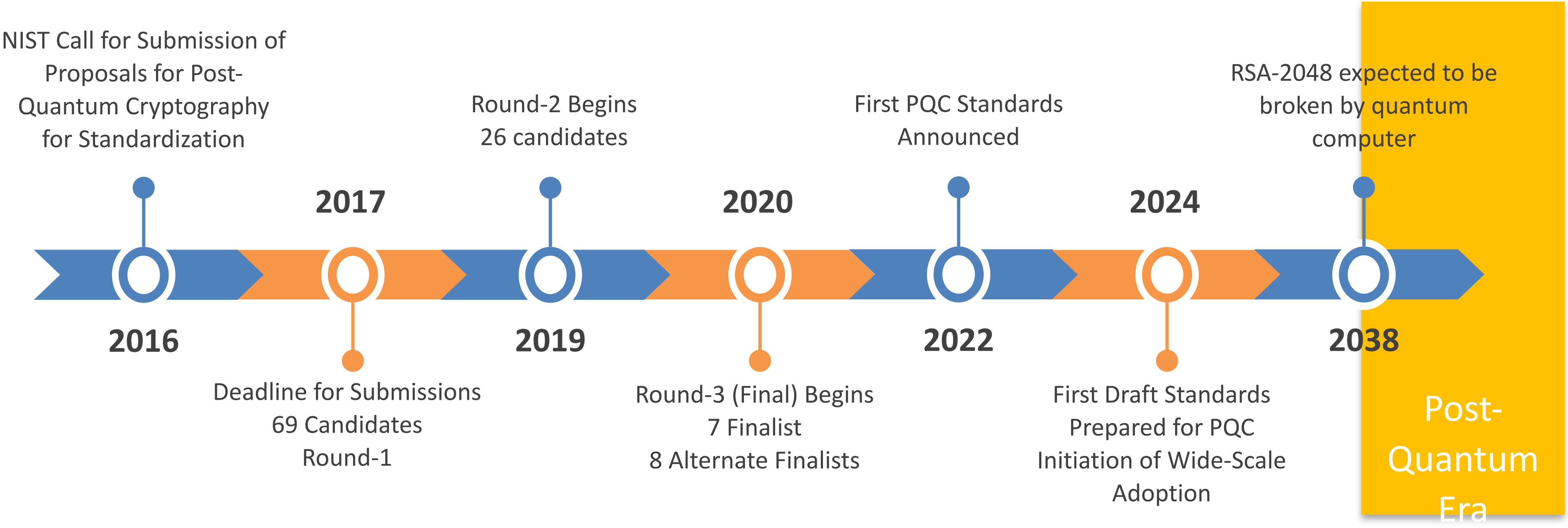


Software Threshold

Table of Contents

1. Introduction to Fault Attacks
2. Persistent Fault Analysis (PFA)
3. PFA on Higher-Order Masking
4. Fault Attack on Lattice based PQC
5. Conclusions

Post-Quantum Cryptography (PQC)



2016

NIST Call for Submission of Proposals for Post-Quantum Cryptography for Standardization

2017

Deadline for Submissions
69 Candidates
Round-1

2019

Round-2 Begins
26 candidates

2020

Round-3 (Final) Begins
7 Finalist
8 Alternate Finalists

2022

First PQC Standards Announced

2024

First Draft Standards Prepared for PQC
Initiation of Wide-Scale Adoption

2038

RSA-2048 expected to be broken by quantum computer

Post-Quantum Era

NIST PQC Standardization

First NIST PQC Standards (US):

PKE/KEMs	Digital Signatures
Kyber (FIPS 203)	Dilithium (FIPS 204)
	FALCON
	SPHINCS+ (FIPS 205)

🟩	Lattice-based
🟧	Hash-based
🟦	Code-based

NIST PQC Standardization

First NIST PQC Standards (US):

PKE/KEMs	Digital Signatures
Kyber (FIPS 203)	Dilithium (FIPS 204)
	FALCON
	SPHINCS+ (FIPS 205)

Round 4 PKE/ KEMs
BIKE
Classic Mceliece
HQC

🟩	Lattice-based
🟧	Hash-based
🟦	Code-based

NIST PQC Standardization

🟩 Lattice-based

🟧 Hash-based

🟦 Code-based

First NIST PQC Standards (US):

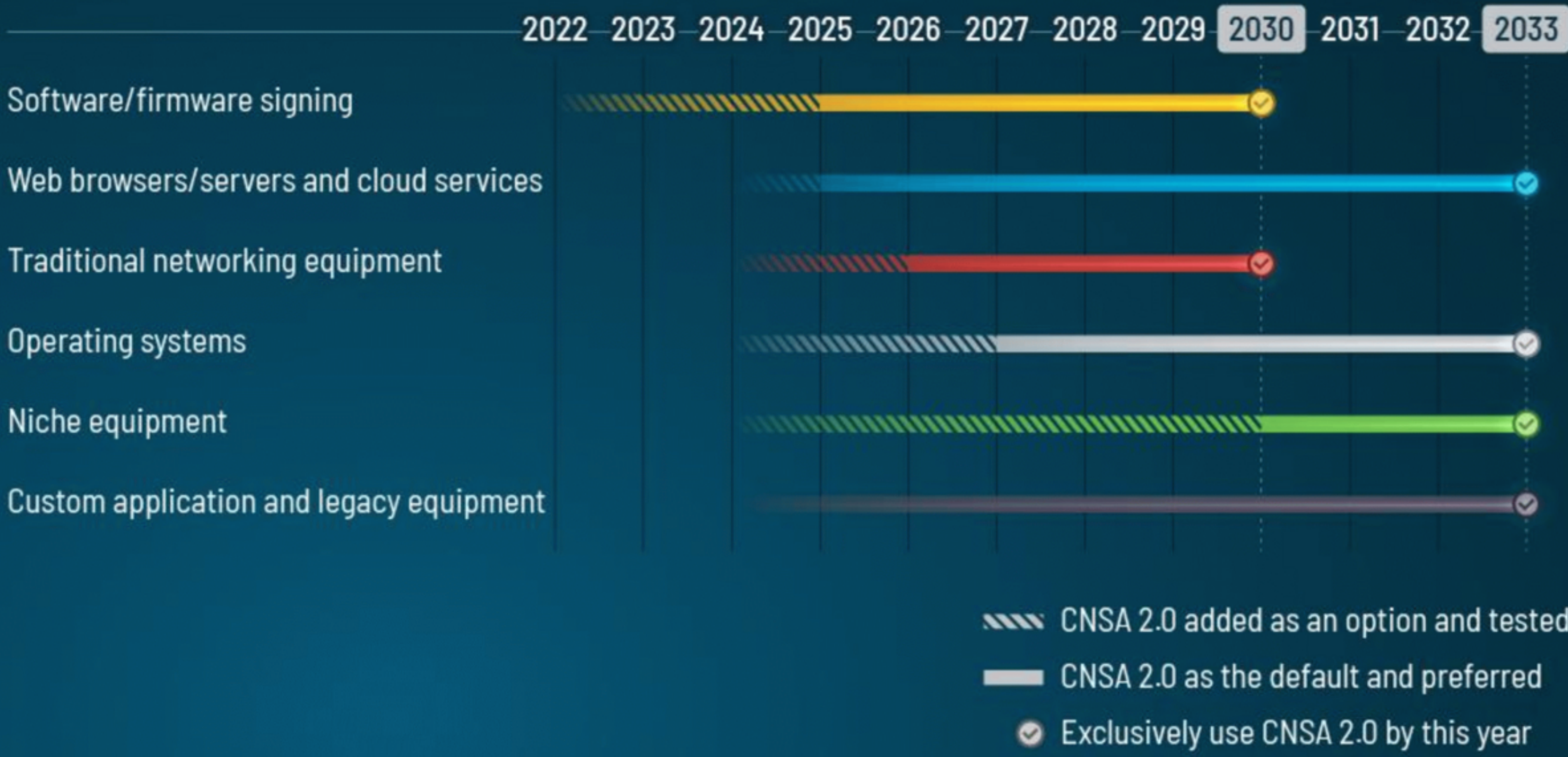
PKE/KEMs	Digital Signatures
Kyber (FIPS 203)	Dilithium (FIPS 204)
	FALCON
	SPHINCS+ (FIPS 205)

Round 4 PKE/ KEMs
BIKE
Classic Mceliece
HQC

BSI Recommendations:

PKE/KEMs	Digital Signatures
FrodoKEM	XMSS
Classic Mceliece	LMS

CNSA 2.0 Timeline

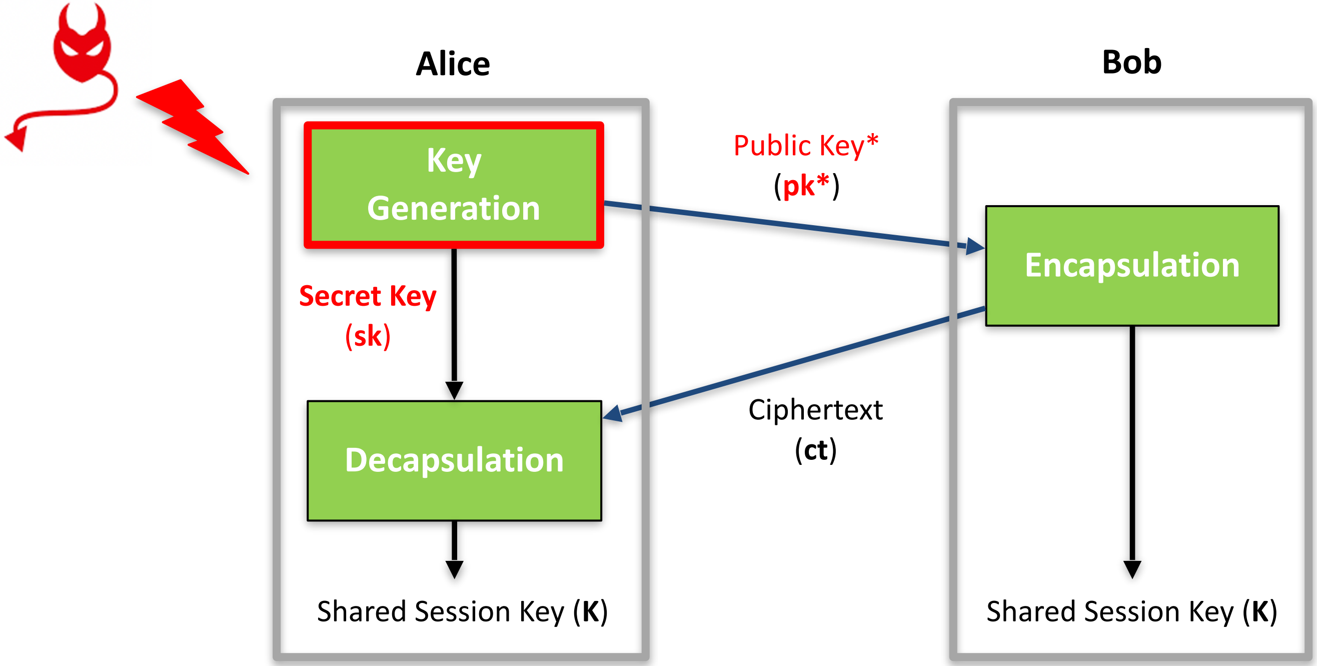


Learning With Error (LWE) Problem

- $\mathbf{T} = (\mathbf{A} * \mathbf{S} + \mathbf{E}) \in Z_q$
  - Secret $\mathbf{S} \in Z_q^n$
  - $\mathbf{A} \in Z_q^n$ is public
  - Error **E** derived from Gaussian distribution

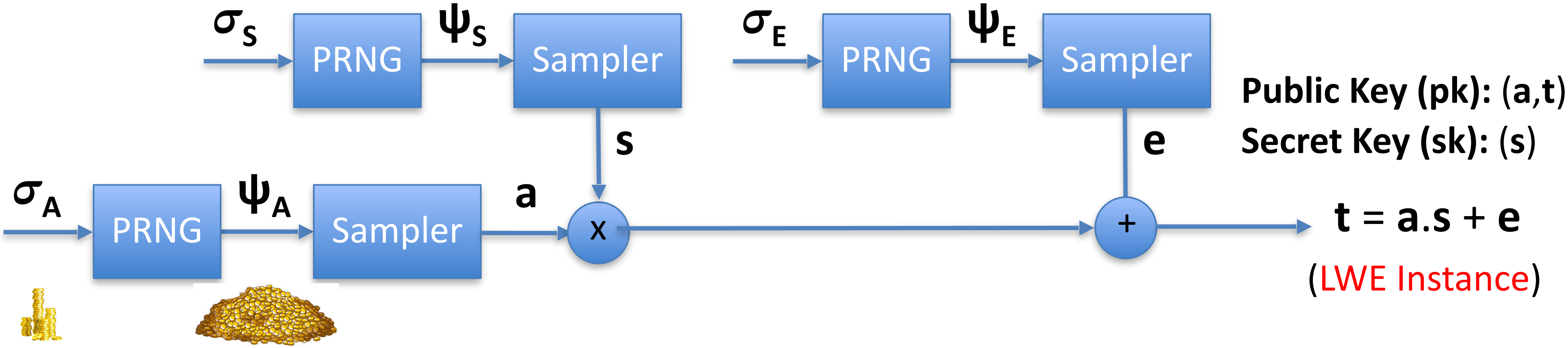
- The hard problem is to solve for **S** given several pairs **(A, T)**
- Error component **E** is essential to hardness guarantees

FIA on Kyber KeyGen



- ❑ Single execution to target Key Generation: Key Recovery Attack
    - ❑ Recover Secret key from Faulty but valid Public Key

FIA on KeyGen: Reduce Entropy of Secret [RYB+23]



σ_S PRNG ψ_S Sampler s

σ_E PRNG ψ_E Sampler e

σ_A PRNG ψ_A Sampler a

x

+

Public Key (pk): (a,t)

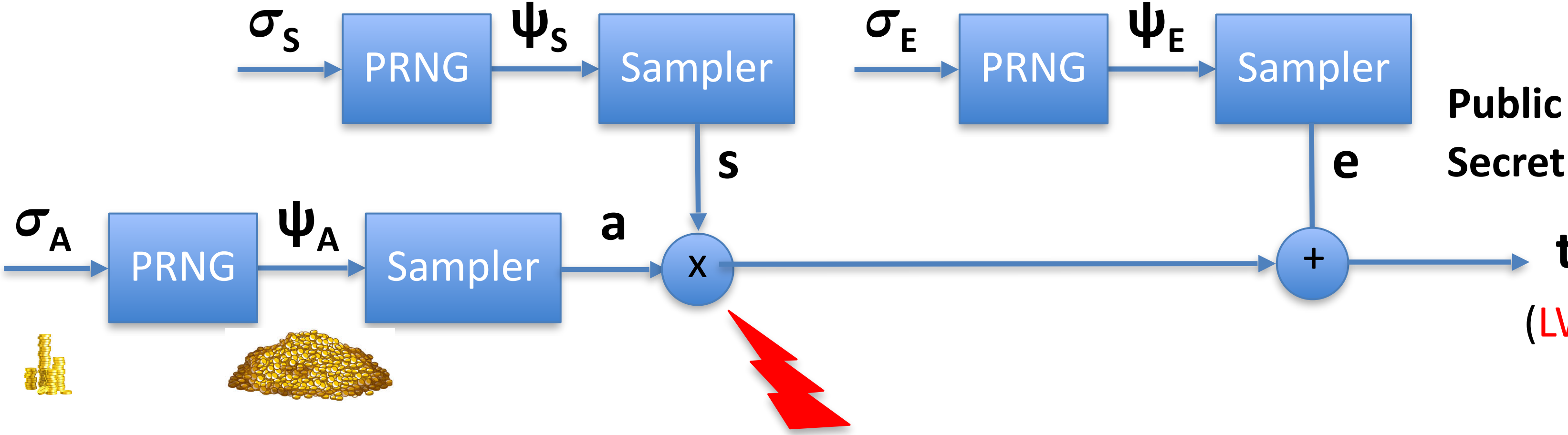
Secret Key (sk): (s)

t = a.s + e

(LWE Instance)

[RYB+23] Ravi, Prasanna, Bolin Yang, Shivam Bhasin, Fan Zhang, and Anupam Chattopadhyay. "Fiddling the Twiddle Constants-Fault Injection Analysis of the Number Theoretic Transform." *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2023): 447-481.

FIA on KeyGen: Reduce Entropy of Secret [RYB+23]



Public Key (pk): (a,t)

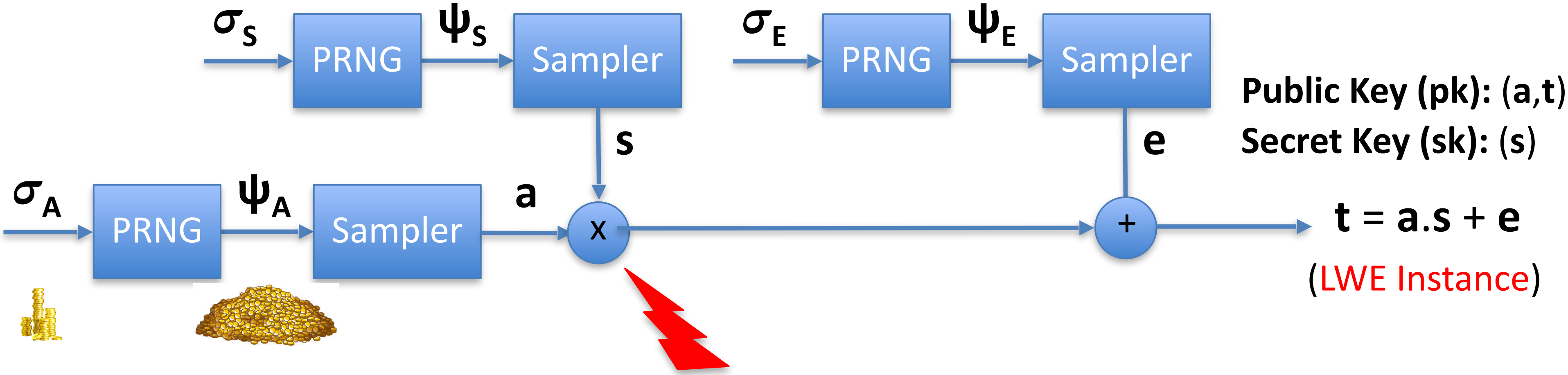
Secret Key (sk): (s)

t = a.s + e

(LWE Instance)

[RYB+23] Ravi, Prasanna, Bolin Yang, Shivam Bhasin, Fan Zhang, and Anupam Chattopadhyay. "Fiddling the Twiddle Constants-Fault Injection Analysis of the Number Theoretic Transform." *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2023): 447-481.

FIA on KeyGen: Reduce Entropy of Secret [RYB+23]



Public Key (pk): (a,t)

Secret Key (sk): (s)

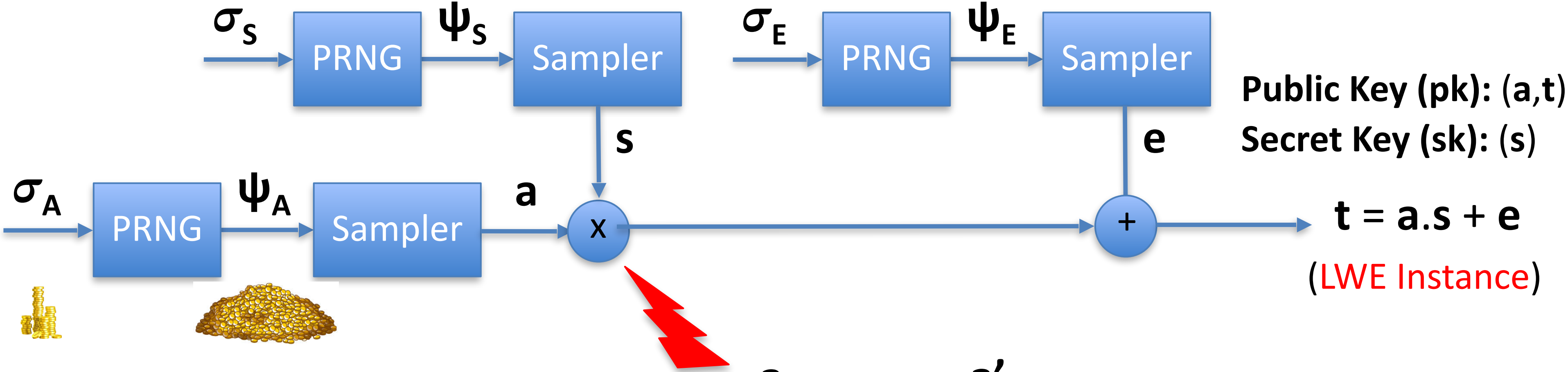
t = a.s + e

(LWE Instance)

Polynomial multiplication is done using Number Theoretic Transform (NTT)

[RYB+23] Ravi, Prasanna, Bolin Yang, Shivam Bhasin, Fan Zhang, and Anupam Chattopadhyay. "Fiddling the Twiddle Constants-Fault Injection Analysis of the Number Theoretic Transform." *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2023): 447-481.

FIA on KeyGen: Reduce Entropy of Secret [RYB+23]

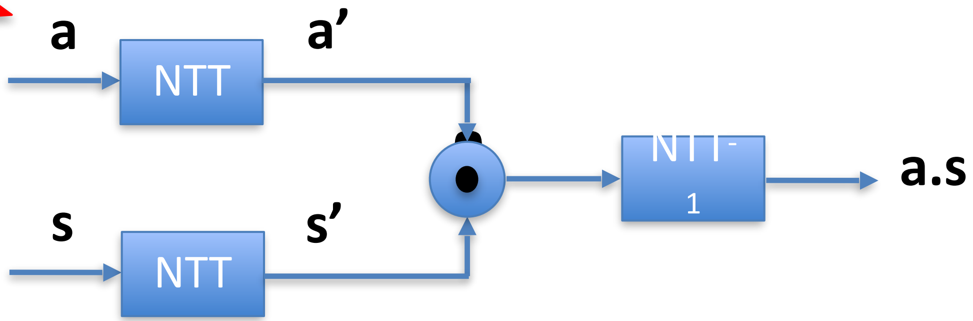


Public Key (pk): (a,t)

Secret Key (sk): (s)

t = a.s + e

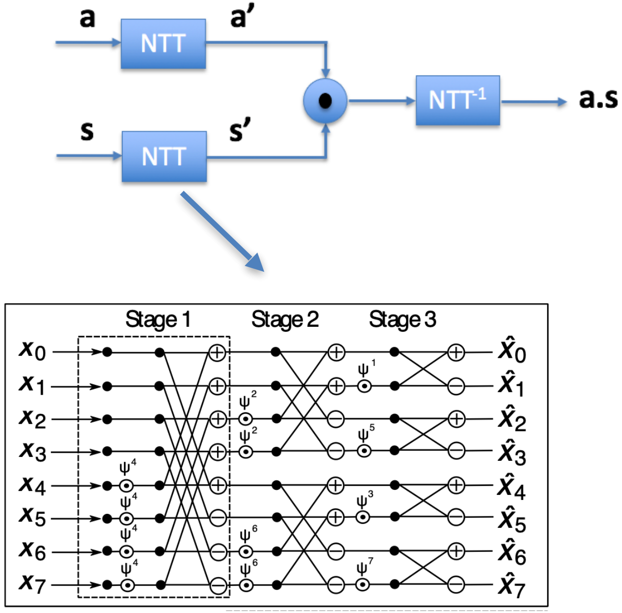
(LWE Instance)



Polynomial multiplication is done using Number Theoretic Transform (NTT)

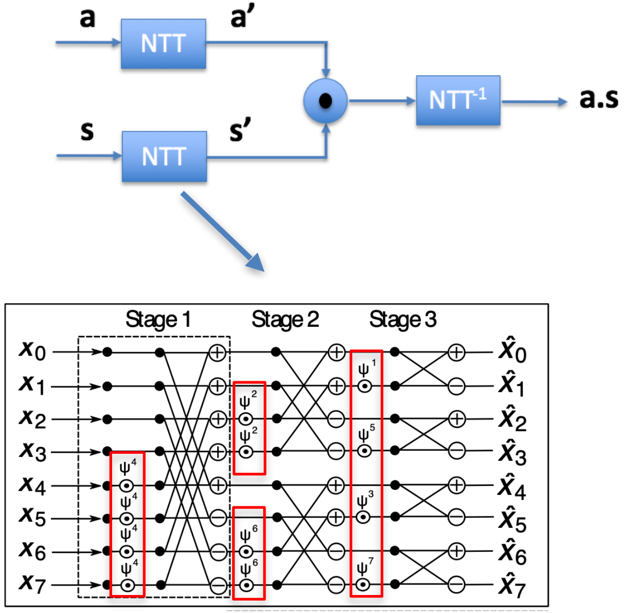
[RYB+23] Ravi, Prasanna, Bolin Yang, Shivam Bhasin, Fan Zhang, and Anupam Chattopadhyay. "Fiddling the Twiddle Constants-Fault Injection Analysis of the Number Theoretic Transform." *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2023): 447-481.

FIA on KeyGen: Reduce Entropy of Secret [RYB+23]



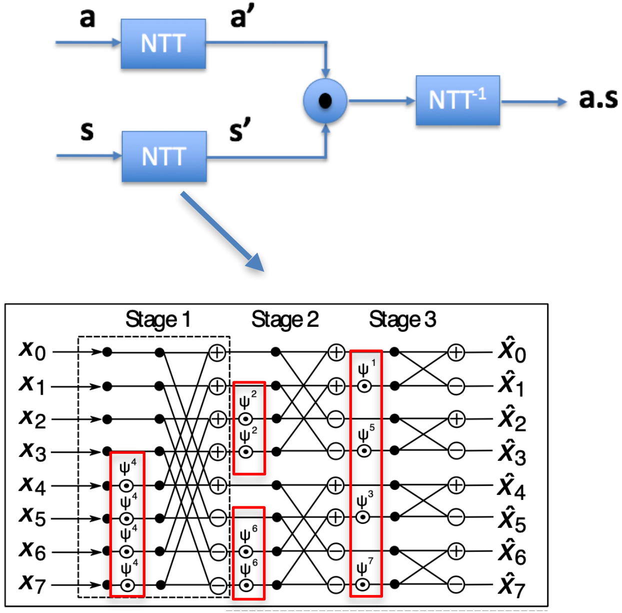
[RYB+22] Ravi, Prasanna, Bolin Yang, Shivam Bhasin, Fan Zhang, and Anupam Chattopadhyay. "Fiddling the Twiddle Constants-Fault Injection Analysis of the Number Theoretic Transform." *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2023): 447-481.

FIA on KeyGen: Reduce Entropy of Secret [RYB+23]



[RYB+22] Ravi, Prasanna, Bolin Yang, Shivam Bhasin, Fan Zhang, and Anupam Chattopadhyay. "Fiddling the Twiddle Constants-Fault Injection Analysis of the Number Theoretic Transform." *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2023): 447-481.

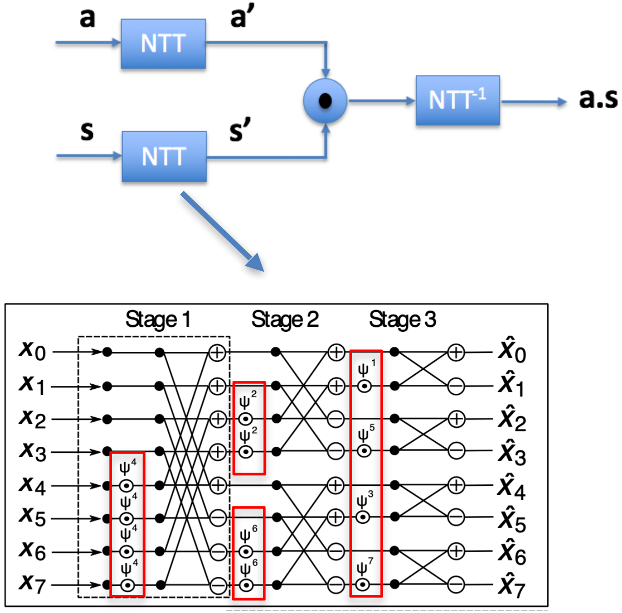
FIA on KeyGen: Reduce Entropy of Secret [RYB+23]



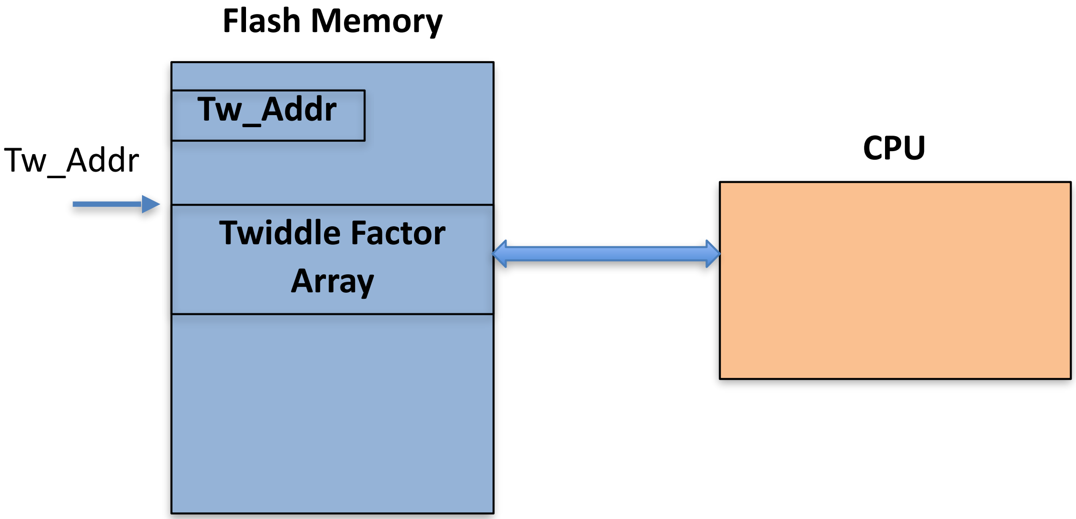
In MCU, Twiddle Constants are stored in Flash Memory as part of Firmware Binary

[RYB+22] Ravi, Prasanna, Bolin Yang, Shivam Bhasin, Fan Zhang, and Anupam Chattopadhyay. "Fiddling the Twiddle Constants-Fault Injection Analysis of the Number Theoretic Transform." *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2023): 447-481.

FIA on KeyGen: Reduce Entropy of Secret [RYB+23]

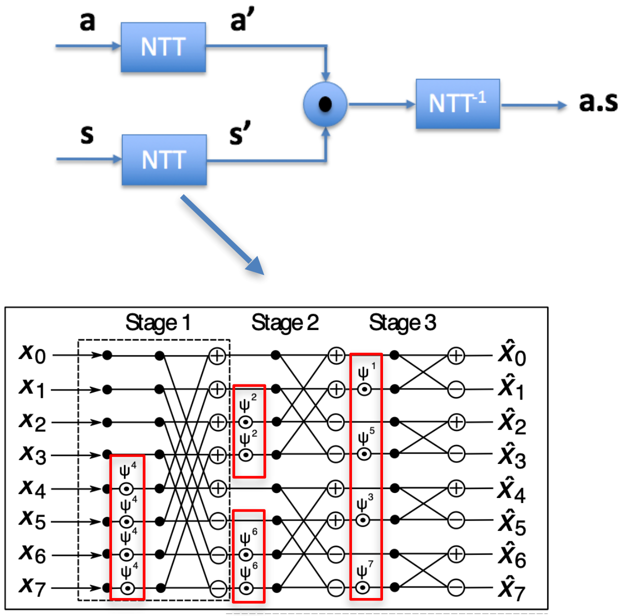


In MCU, Twiddle Constants are stored in Flash Memory as part of Firmware Binary

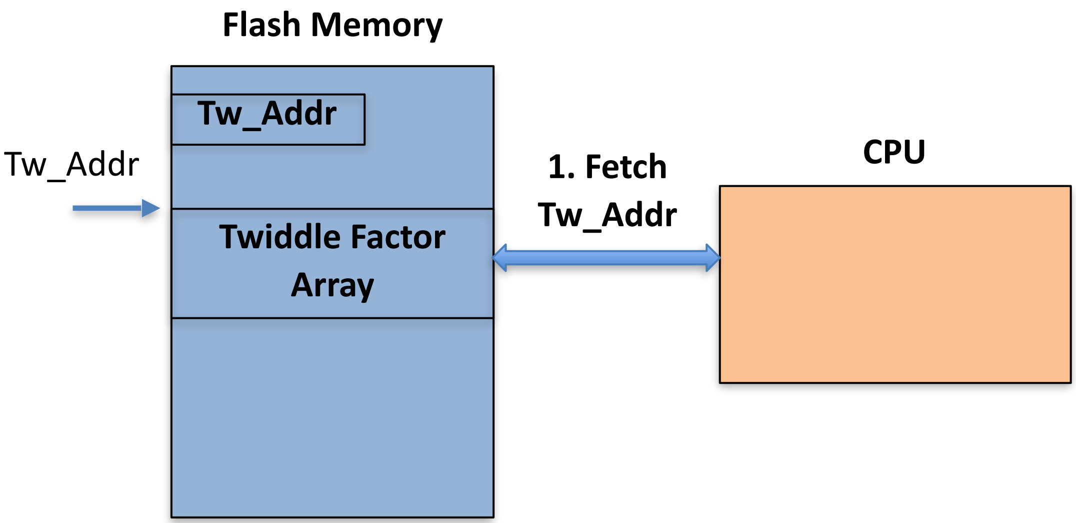


[RYB+22] Ravi, Prasanna, Bolin Yang, Shivam Bhasin, Fan Zhang, and Anupam Chattopadhyay. "Fiddling the Twiddle Constants-Fault Injection Analysis of the Number Theoretic Transform." *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2023): 447-481.

FIA on KeyGen: Reduce Entropy of Secret [RYB+23]

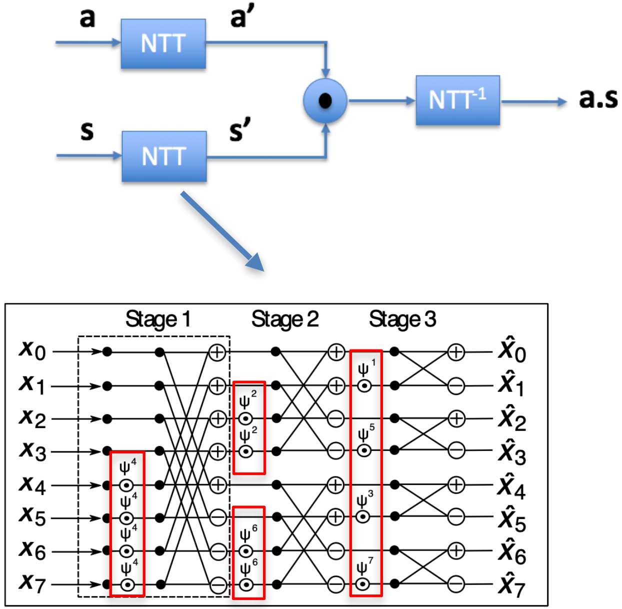


In MCU, Twiddle Constants are stored in Flash Memory as part of Firmware Binary

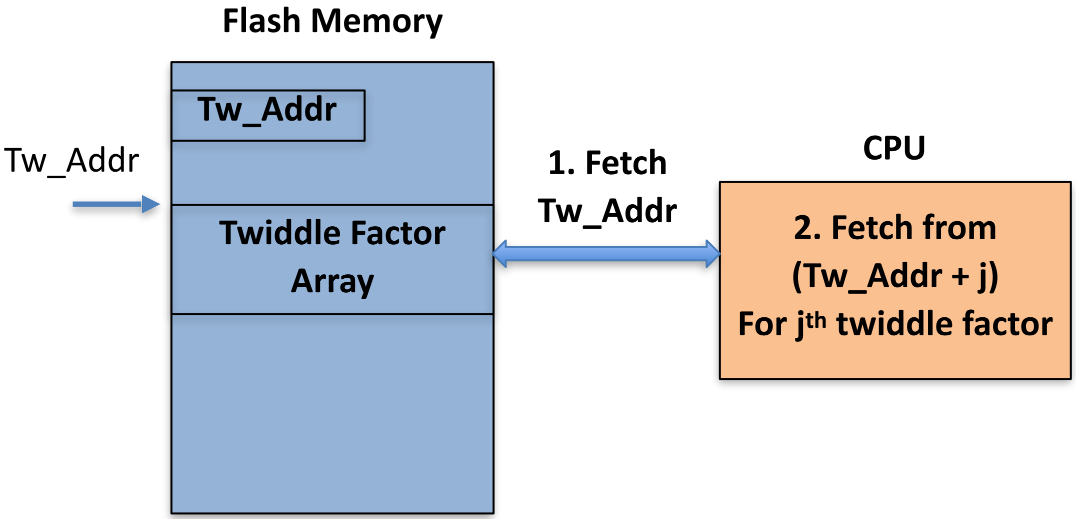


[RYB+22] Ravi, Prasanna, Bolin Yang, Shivam Bhasin, Fan Zhang, and Anupam Chattopadhyay. "Fiddling the Twiddle Constants-Fault Injection Analysis of the Number Theoretic Transform." *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2023): 447-481.

FIA on KeyGen: Reduce Entropy of Secret [RYB+23]

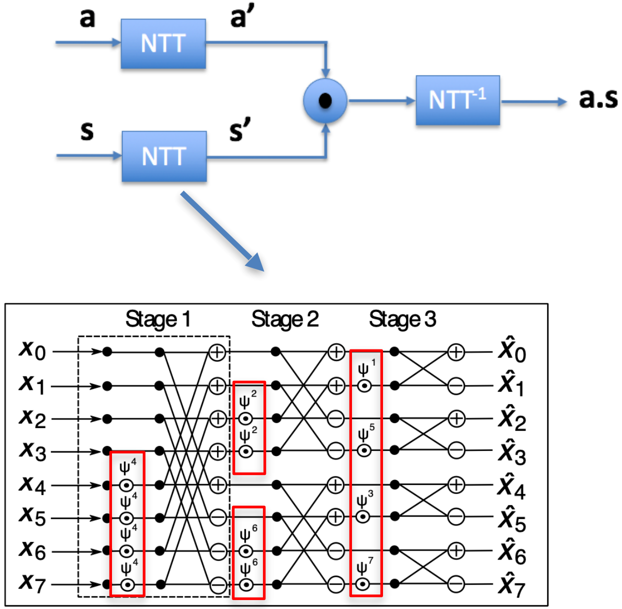


In MCU, Twiddle Constants are stored in Flash Memory as part of Firmware Binary

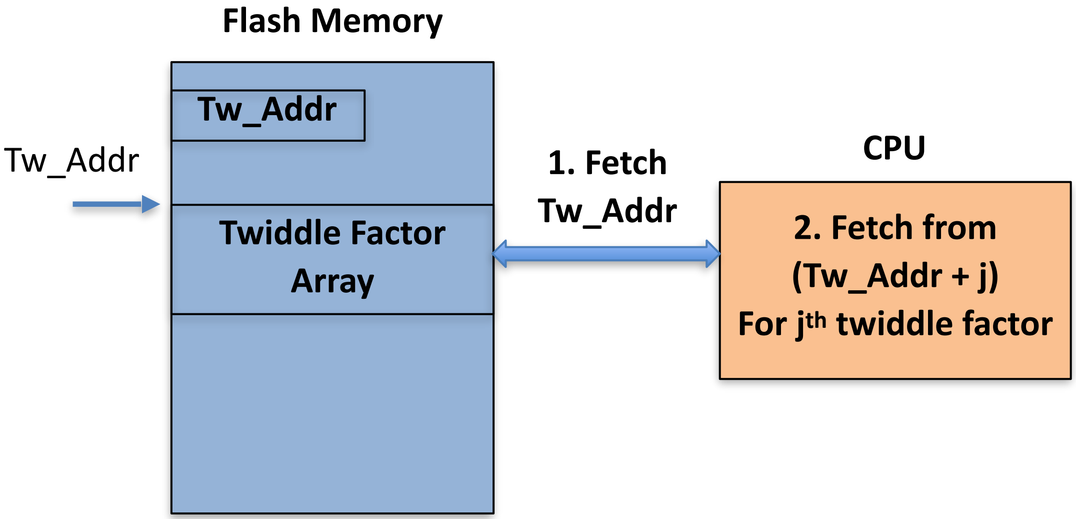


[RYB+22] Ravi, Prasanna, Bolin Yang, Shivam Bhasin, Fan Zhang, and Anupam Chattopadhyay. "Fiddling the Twiddle Constants-Fault Injection Analysis of the Number Theoretic Transform." *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2023): 447-481.

FIA on KeyGen: Reduce Entropy of Secret [RYB+23]



In MCU, Twiddle Constants are stored in Flash Memory as part of Firmware Binary



Flash Memory

Tw_Addr

Tw_Addr

Twiddle Factor Array

1. Fetch Tw_Addr

CPU

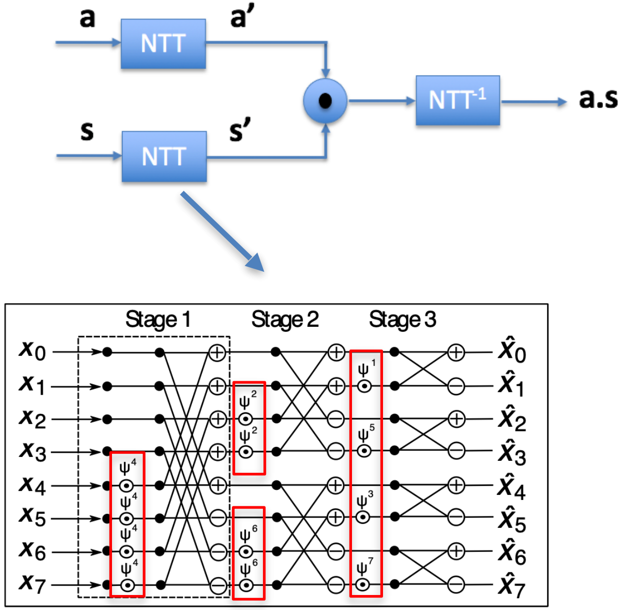
2. Fetch from (Tw_Addr + j) For jth twiddle factor

Main Observation: Tw_Addr is used as **base-address** to calculate address for all constants

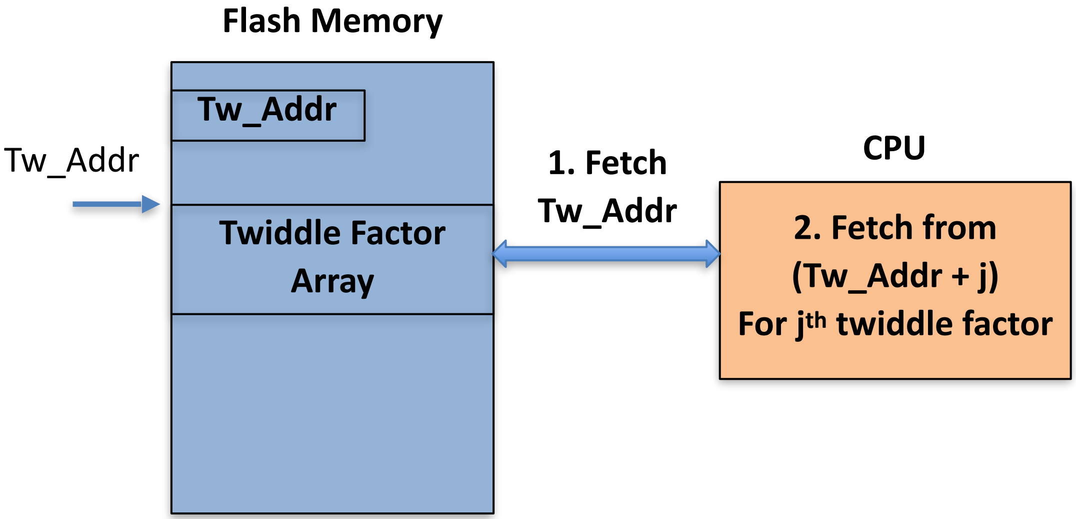
Fault Vulnerability: Can an attacker fault the base address?

[RYB+22] Ravi, Prasanna, Bolin Yang, Shivam Bhasin, Fan Zhang, and Anupam Chattopadhyay. "Fiddling the Twiddle Constants-Fault Injection Analysis of the Number Theoretic Transform." *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2023): 447-481.

FIA on KeyGen: Reduce Entropy of Secret [RYB+23]



In MCU, Twiddle Constants are stored in Flash Memory as part of Firmware Binary



Flash Memory

Tw_Addr

Tw_Addr

Twiddle Factor Array

1. Fetch Tw_Addr

CPU

2. Fetch from (Tw_Addr + j) For j^{th} twiddle factor

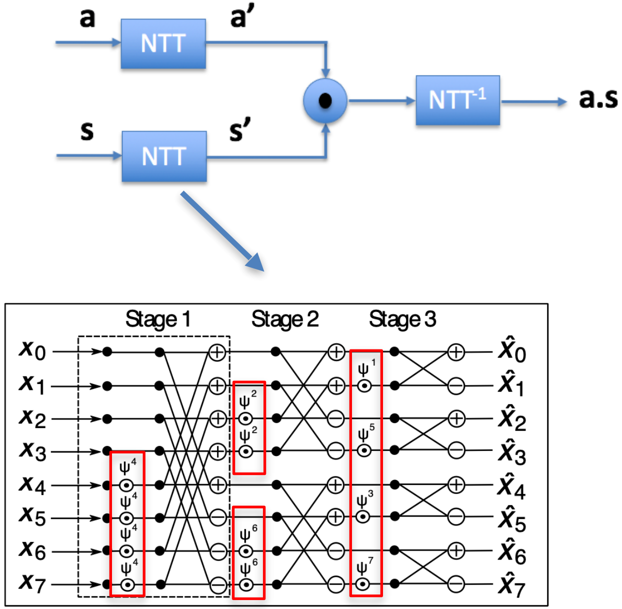
Main Observation: Tw_Addr is used as **base-address** to calculate address for all constants

Fault Vulnerability: Can an attacker fault the base address?

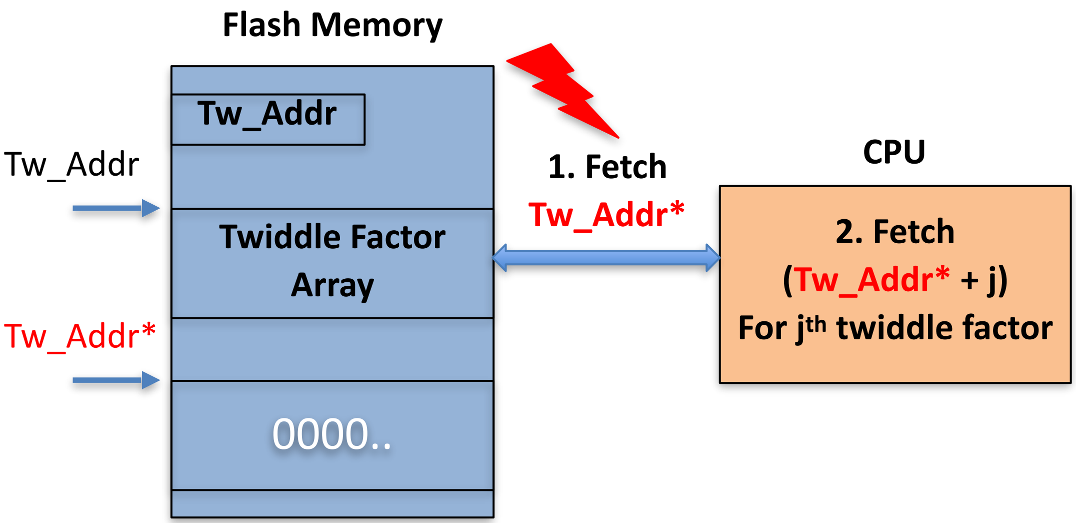
Implementation Style used in all publicly available optimized implementations of Kyber and Dilithium for ARM Cortex-M4 Processor

[RYB+22] Ravi, Prasanna, Bolin Yang, Shivam Bhasin, Fan Zhang, and Anupam Chattopadhyay. "Fiddling the Twiddle Constants-Fault Injection Analysis of the Number Theoretic Transform." *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2023): 447-481.

FIA on KeyGen: Reduce Entropy of Secret [RYB+23]



In MCU, Twiddle Constants are stored in Flash Memory as part of Firmware Binary



Flash Memory

Tw_Addr

Tw_Addr

Twiddle Factor Array

Tw_Addr*

0000..

1. Fetch Tw_Addr*

CPU

2. Fetch (Tw_Addr* + j) For jth twiddle factor

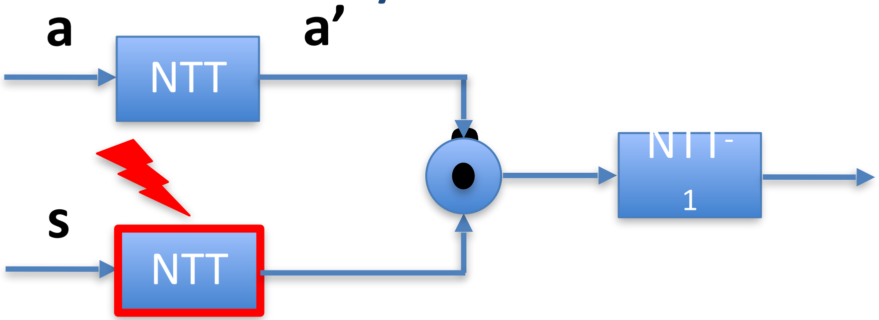
Observation: Can zeroize the entire twiddle factor array in a single fault

25% of random memory locations yield zeros on ARM Cortex-M4 processor

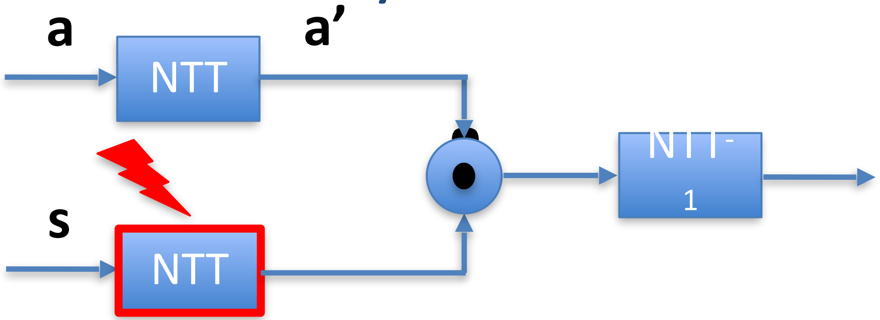
What happens when twiddle factors are zeroized???

[RYB+22] Ravi, Prasanna, Bolin Yang, Shivam Bhasin, Fan Zhang, and Anupam Chattopadhyay. "Fiddling the Twiddle Constants-Fault Injection Analysis of the Number Theoretic Transform." *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2023): 447-481.

NTT Fault Vulnerability: Zeroization of Twiddle Constants

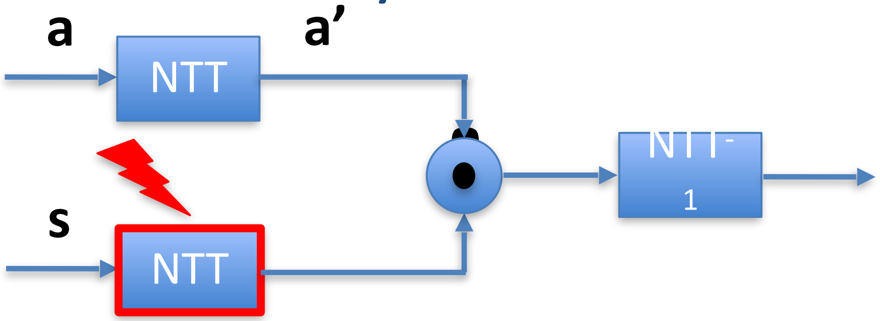


NTT Fault Vulnerability: Zeroization of Twiddle Constants



Entropy reduces by half in every NTT layer

NTT Fault Vulnerability: Zeroization of Twiddle Constants

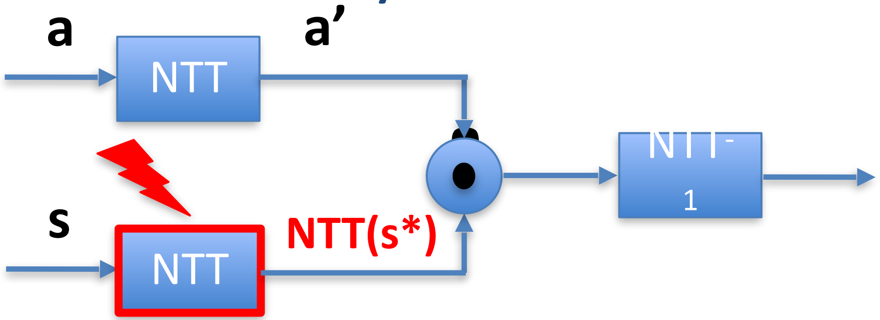


Entropy reduces by half in every NTT layer

s0	s1	s2	s3	s4	s5	s6	s7

s0	s1	s0	s1	s0	s1	s0	s1

NTT Fault Vulnerability: Zeroization of Twiddle Constants

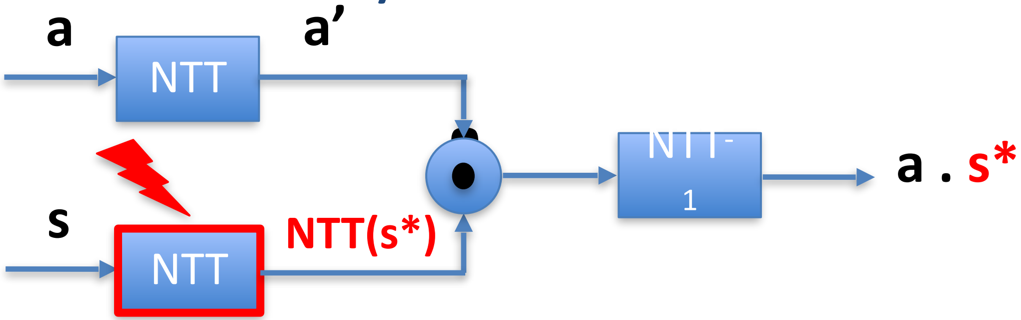


Entropy reduces by half in every NTT layer

s0	s1	s2	s3	s4	s5	s6	s7

s0	s1	s0	s1	s0	s1	s0	s1

NTT Fault Vulnerability: Zeroization of Twiddle Constants

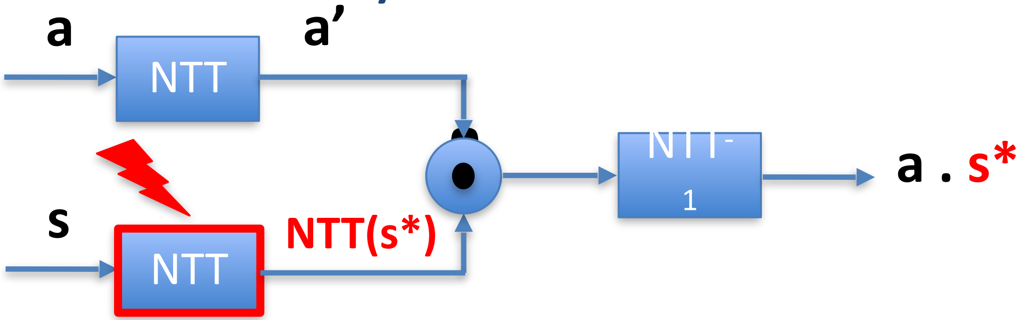


Entropy reduces by half in every NTT layer

s0	s1	s2	s3	s4	s5	s6	s7

s0	s1	s0	s1	s0	s1	s0	s1

NTT Fault Vulnerability: Zeroization of Twiddle Constants



Faulty secret s*

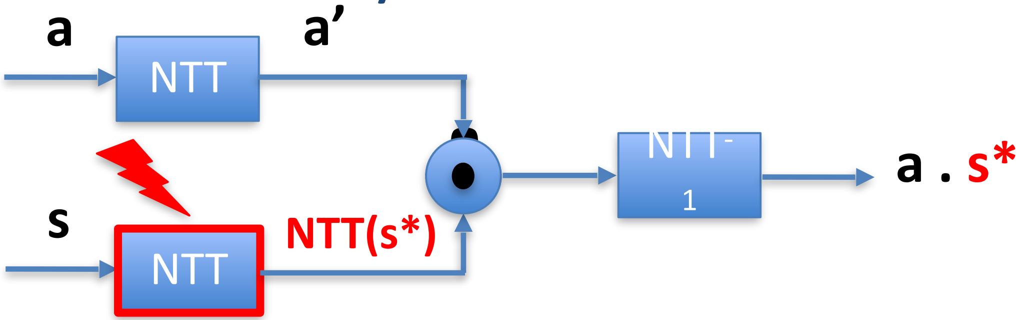
s0	s1	0	0	0	0	0	0

Entropy reduces by half in every NTT layer

s0	s1	s2	s3	s4	s5	s6	s7

s0	s1	s0	s1	s0	s1	s0	s1

NTT Fault Vulnerability: Zeroization of Twiddle Constants



Valid Secret, but with
Low Entropy

Faulty secret s*

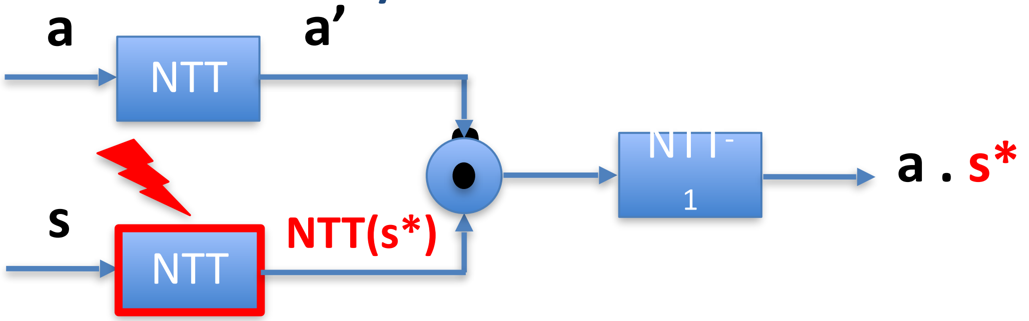
s0	s1	0	0	0	0	0	0

Entropy reduces by half in every NTT layer

s0	s1	s2	s3	s4	s5	s6	s7

s0	s1	s0	s1	s0	s1	s0	s1

NTT Fault Vulnerability: Zeroization of Twiddle Constants



Valid Secret, but with Low Entropy

Faulty secret s*

s0	s1	0	0	0	0	0	0

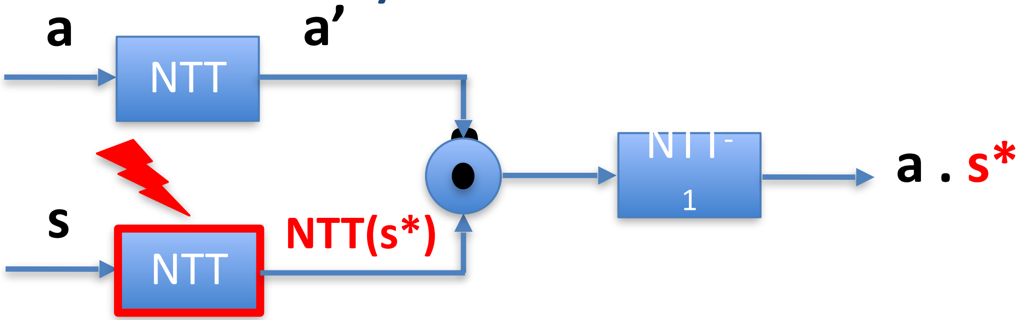
Entropy reduces by half in every NTT layer

s0	s1	s2	s3	s4	s5	s6	s7

s0	s1	s0	s1	s0	s1	s0	s1

- ❑ Kyber uses Incomplete NTT
  - ❑ 7 layers (256 point NTT)
  - ❑ Two non zero coeff. at NTT output
- ❑ Dilithium uses complete NTT
  - ❑ 8 layers (256 point NTT)
  - ❑ One non-zero coeff. At NTT output

NTT Fault Vulnerability: Zeroization of Twiddle Constants



Valid Secret, but with Low Entropy

Entropy reduces by half in every NTT layer

s0	s1	s2	s3	s4	s5	s6	s7

s0	s1	s0	s1	s0	s1	s0	s1

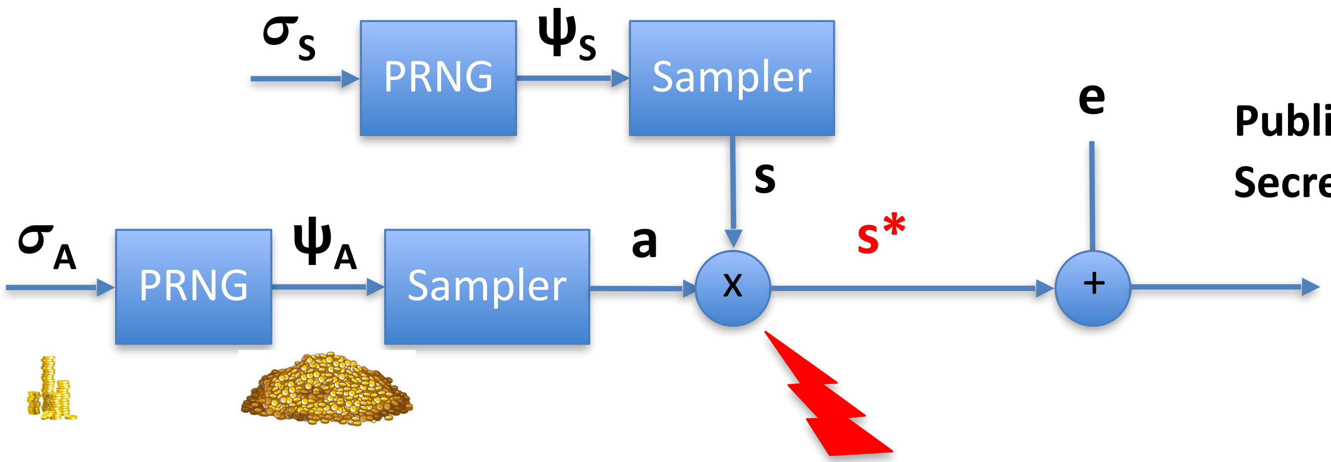
Faulty secret s*

s0	s1	0	0	0	0	0	0

- Secret **s** has **k** polynomials
  - **k** NTTs
- But, we experimentally observed that fault on one NTT is sufficient
- Maybe faulty twiddle pointer is cached and reused for **k** NTTs

- Kyber uses Incomplete NTT
  - 7 layers (256 point NTT)
  - Two non zero coeff. at NTT output
- Dilithium uses complete NTT
  - 8 layers (256 point NTT)
  - One non-zero coeff. At NTT output

FIA on Kyber KeyGen: Zeroization of Twiddle Constants

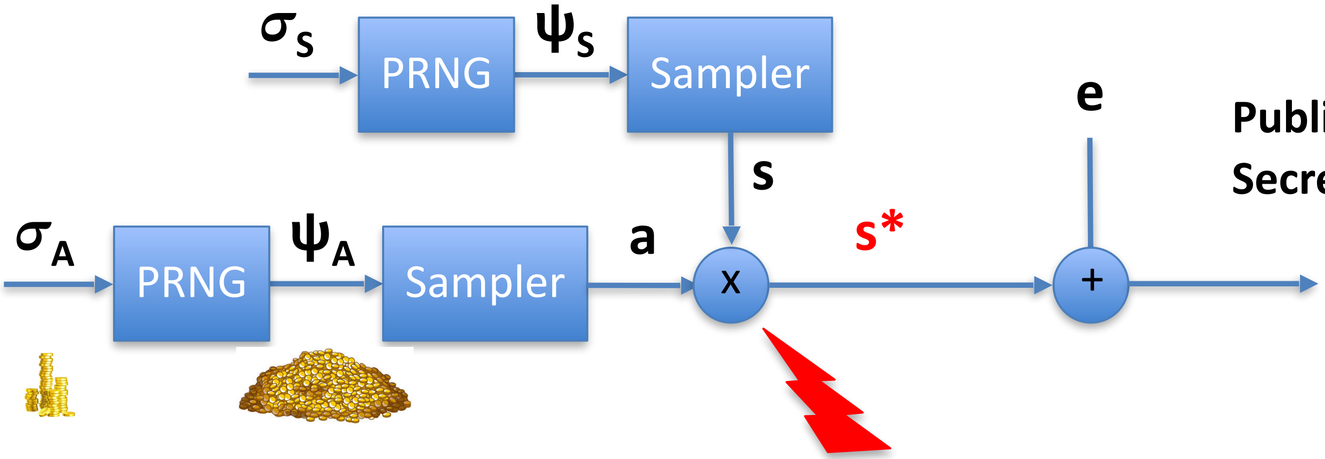


$t^* = a.s^* + e$

Public Key (pk): (a,t)

Secret Key (sk): (s^*)

FIA on Kyber KeyGen: Zeroization of Twiddle Constants



Public Key (pk): (**a**,**t**)

Secret Key (sk): (**s***)

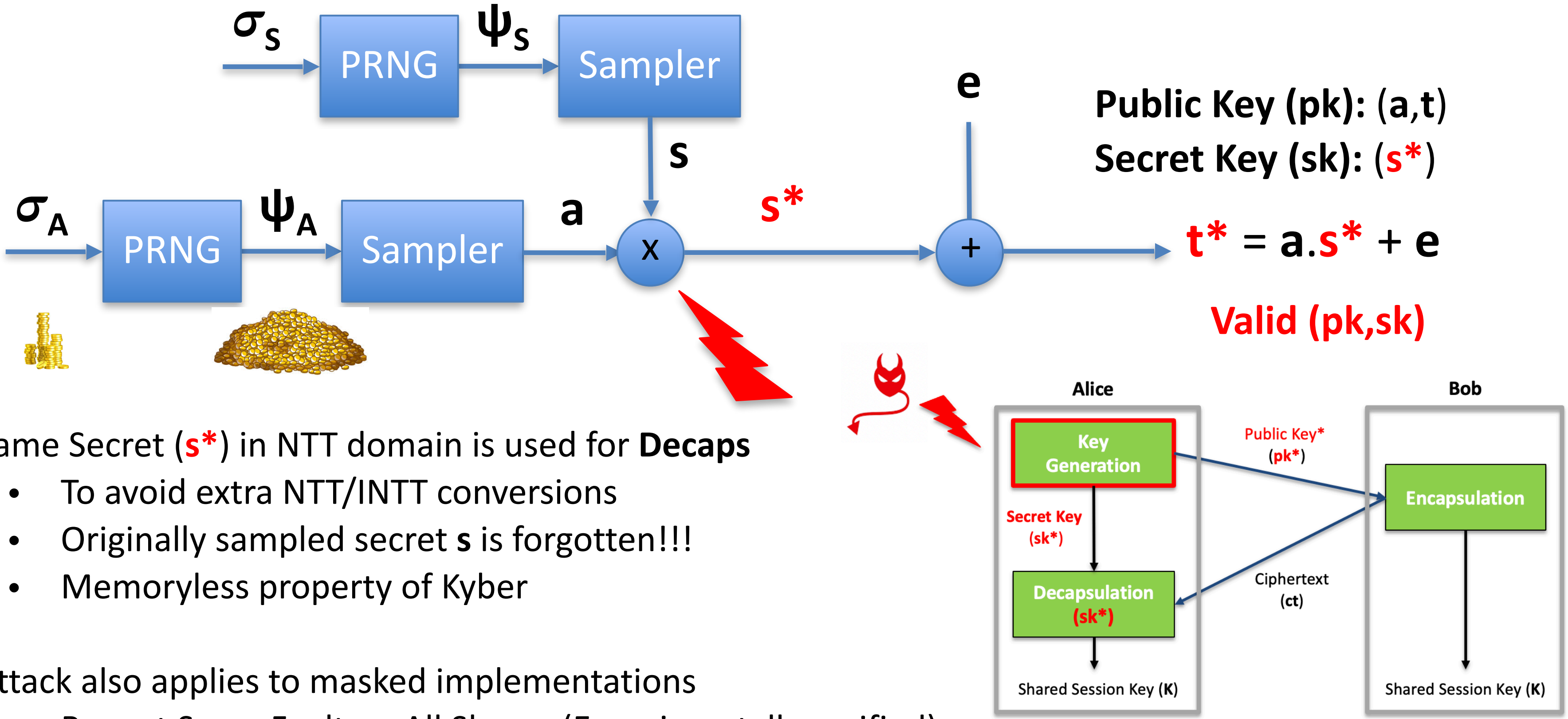
t* = **a**.**s*** + **e**

Valid (pk,sk)

- Same Secret (**s***) in NTT domain is used for **Decaps**
  - To avoid extra NTT/INTT conversions
  - Originally sampled secret **s** is forgotten!!!
  - Memoryless property of Kyber

- Attack also applies to masked implementations
  - Repeat Same Fault on All Shares (Experimentally verified)

FIA on Kyber KeyGen: Zeroization of Twiddle Constants



Public Key (pk): (a,t)

Secret Key (sk): (s*)

$t^* = a.s^* + e$

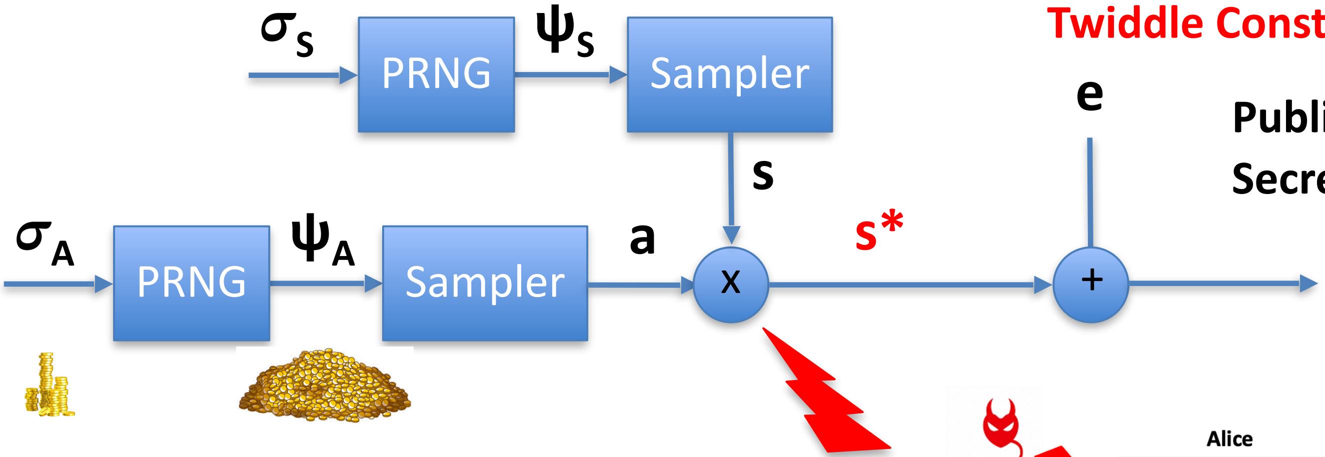
Valid (pk,sk)

- Same Secret (**s***) in NTT domain is used for **Decaps**
  - To avoid extra NTT/INTT conversions
  - Originally sampled secret **s** is forgotten!!!
  - Memoryless property of Kyber

- Attack also applies to masked implementations
  - Repeat Same Fault on All Shares (Experimentally verified)

FIA on Kyber KeyGen: Zeroization of Twiddle Constants

Countermeasure: Sanity Check on Twiddle Constants or NTT outputs



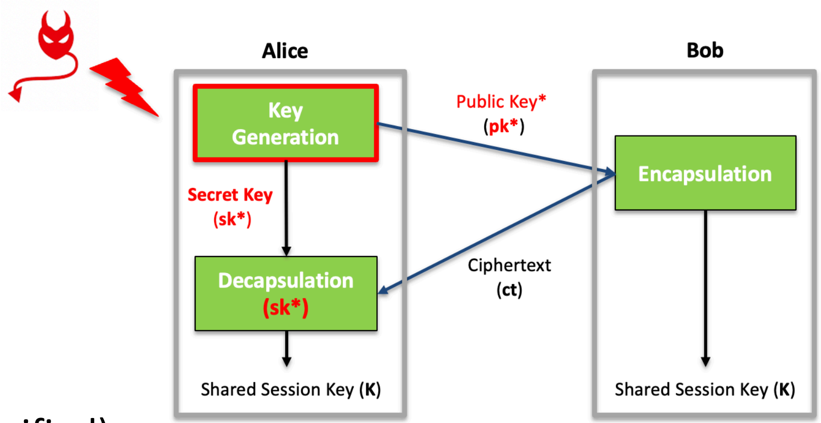
Public Key (pk): (a,t)

Secret Key (sk): (s*)

$$t^* = a.s^* + e$$

Valid (pk,sk)

- Same Secret (s*) in NTT domain is used for **Decaps**
  - To avoid extra NTT/INTT conversions
  - Originally sampled secret **s** is forgotten!!!
  - Memoryless property of Kyber



- Attack also applies to masked implementations
  - Repeat Same Fault on All Shares (Experimentally verified)

Table of Contents

1. Introduction to Fault Attacks
2. Persistent Fault Analysis (PFA)
3. PFA on Higher-Order Masking
4. Fault Attack on Lattice based PQC
5. Conclusions

Conclusions

- Faults attack are a powerful attack vector
- With good control over setup, even a single fault can be devastating
- Demonstrated the power in context of block ciphers, protection mechanisms, PQC etc
- A study of fault injection capabilities and fault analysis must go hand in hand
- A lot is still left to explore
- Are we moving towards formal analysis of security analysis against fault attacks and combined attacks?

Thank You !!!