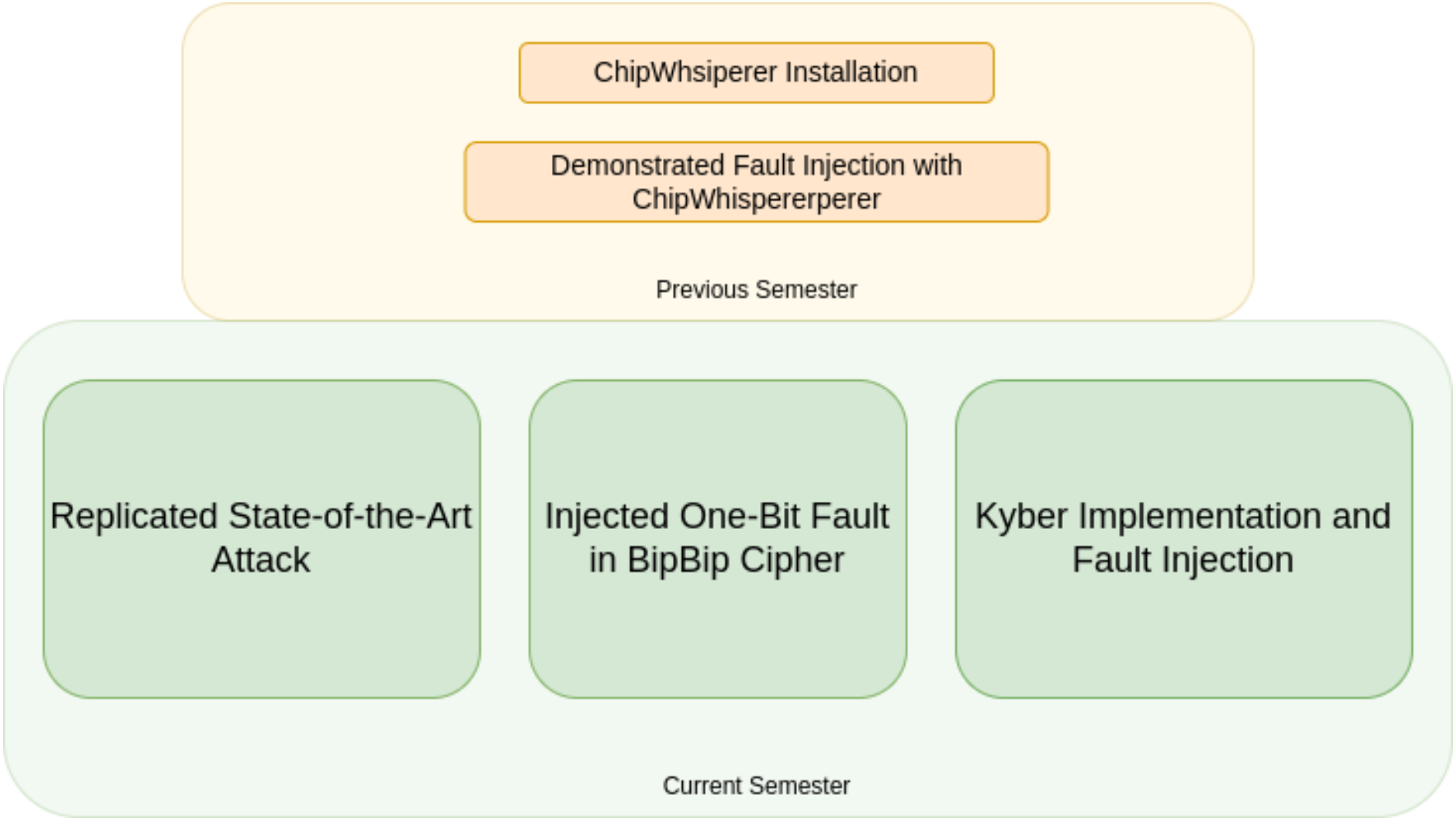
Injecting Chaos: Fault Attack Setup and Analysis Via ChipWhisperer

Chayan Pathak
12310630
M.Tech CSE
Supervisor: Dr. Dhiman Saha

Research Timeline



Contents

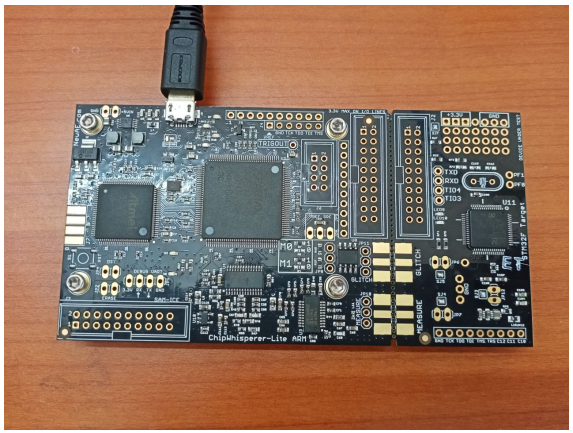
Introduction To Fault Attacks

- **Fault Injection Attacks (FIA)** are hardware-based attacks that introduce faults into a system to alter its behavior.
- Attackers induce transient errors to bypass security checks, alter data or extract sensitive data.
- Common techniques:
  - Voltage Glitching
  - Clock Glitching
  - Electromagnetic (EM) Pulses
  - Laser Fault Injection
- Often used to target cryptographic algorithms (e.g., AES, RSA).

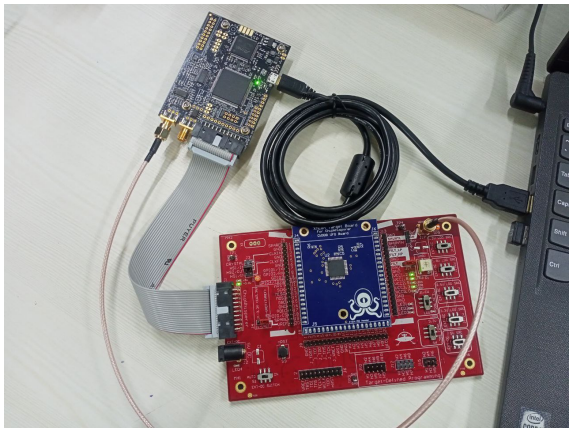
Setting Up a Fault Injection Lab with ChipWhisperer

- **ChipWhisperer** is an open-source toolchain for side-channel and fault injection research.
- Designed for teaching reasearch and evaluating hardware security.
- Includes:
  - CWNano,CWLite or CWHusky hardware for capturing traces.
  - Target board (e.g., XMEGA, STM32, SAM4S).
  - Software suite for glitching, capturing, and analysis.
- Supports fault injection via:
  - Clock Glitching
  - Voltage Faults
  - Electromagnetic Pulse Injection (with add-ons like ChipShouter)

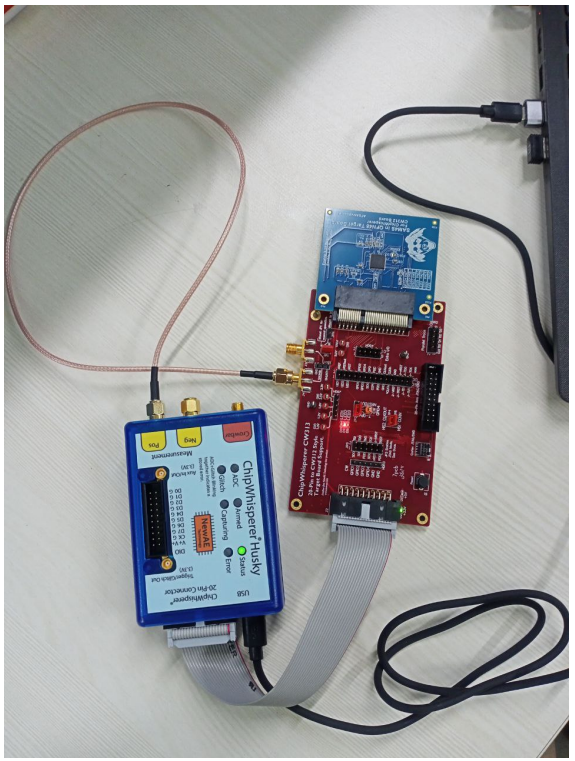
Hardware Setup



CWNLite with ARM 32-bit Target



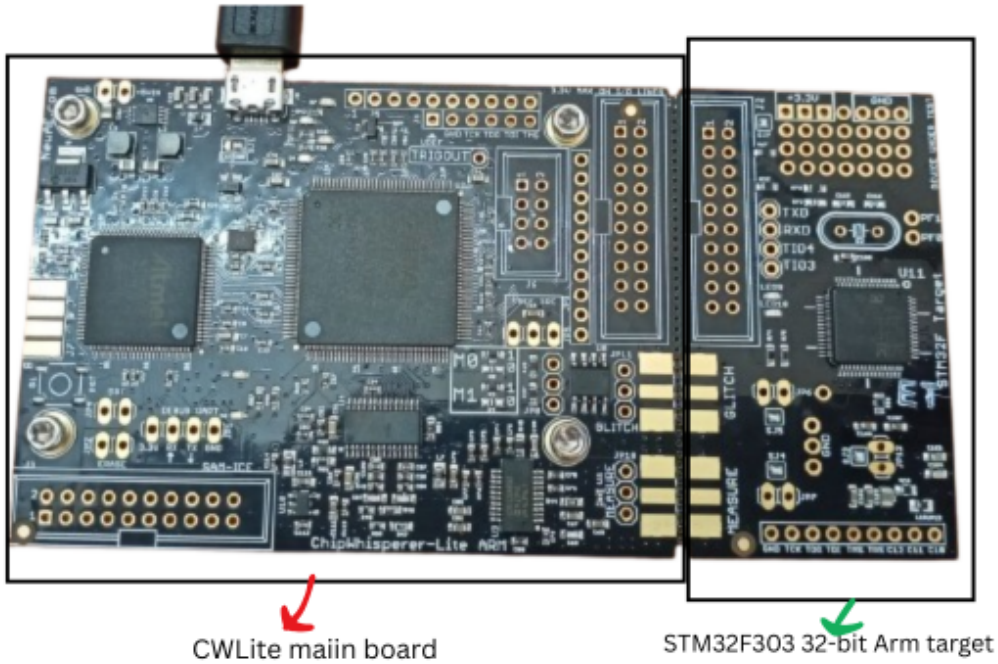
CWLite with Separate Target



CWHusky setup

CWLite with ARM 32-bit Target

- 10-bit 105MS/s ADC for capturing power traces
- Clock and voltage fault generation via FPGA-based pulse generation
- Sample Buffer Size: 24,573 samples
- Capable of doing both Voltage glitching and clock glitching



Software Environment

Real-world Replication of the State-of-the-Art

Diagonals in the AES

Diagonal 0 $[D_0]$

a_{00}	a_{01}	a_{02}	a_{03}
a_{10}	a_{11}	a_{12}	a_{13}
a_{20}	a_{21}	a_{22}	a_{23}
a_{30}	a_{31}	a_{32}	a_{33}

Diagonal 1 $[D_1]$

a_{00}	a_{01}	a_{02}	a_{03}
a_{10}	a_{11}	a_{12}	a_{13}
a_{20}	a_{21}	a_{22}	a_{23}
a_{30}	a_{31}	a_{32}	a_{33}

Diagonal 2 $[D_2]$

a_{00}	a_{01}	a_{02}	a_{03}
a_{10}	a_{11}	a_{12}	a_{13}
a_{20}	a_{21}	a_{22}	a_{23}
a_{30}	a_{31}	a_{32}	a_{33}

Diagonal 3 $[D_3]$

a_{00}	a_{01}	a_{02}	a_{03}
a_{10}	a_{11}	a_{12}	a_{13}
a_{20}	a_{21}	a_{22}	a_{23}
a_{30}	a_{31}	a_{32}	a_{33}

Fault Injection at the Start of Round 8 using CW-Lite

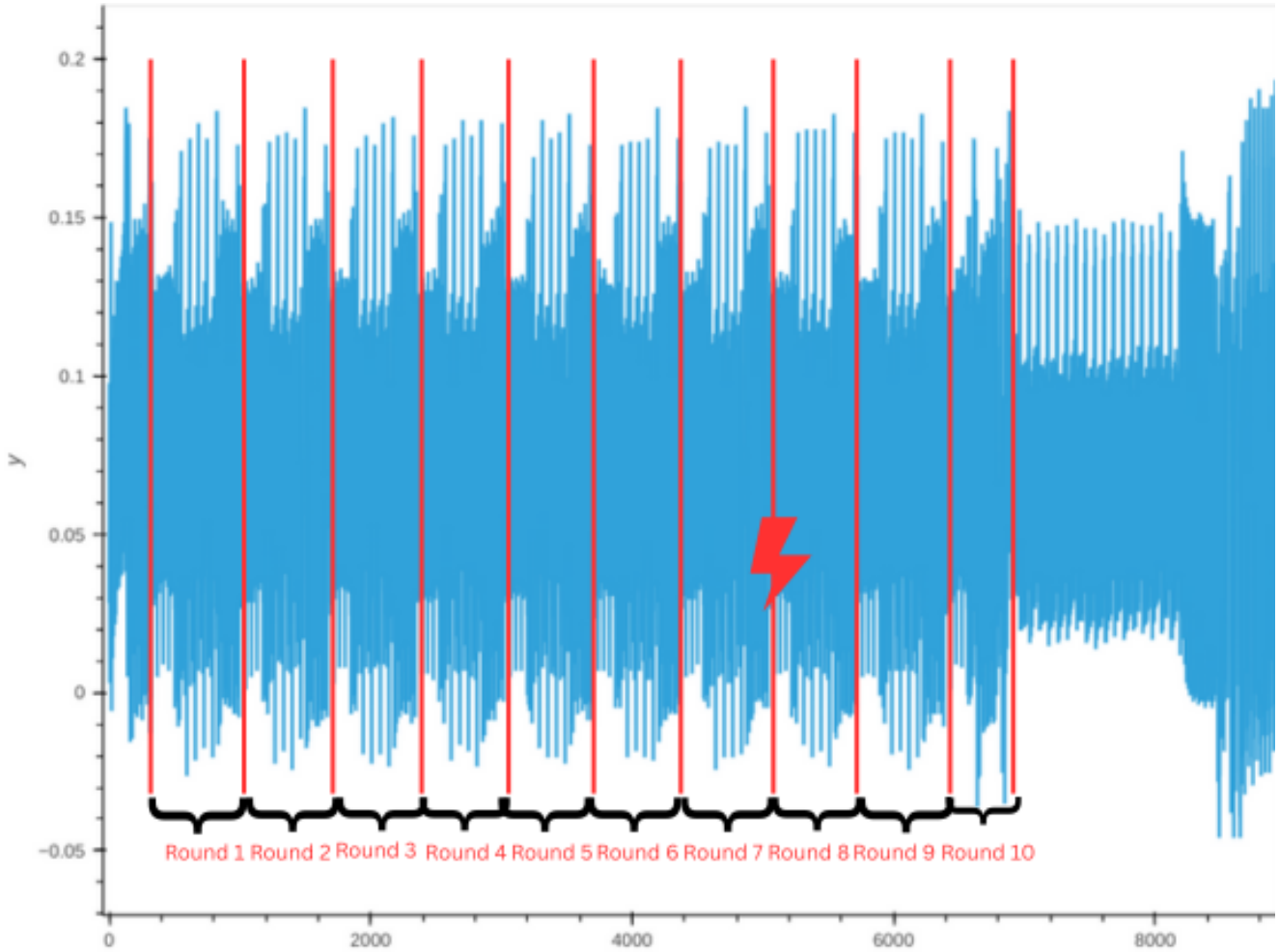


Figure: Power Trace of AES128

Using Clock Glitching

Target: AES-128 on CWLite-ARM

- 8th round identified at sample range: **5080–5120**
- Clock glitch injected at: **5086 samples**

Glitch Configuration:

```
scope.glitch.clk_src = "clkgen"
scope.glitch.output = "clock_xor"
scope.glitch.trigger_src = "ext_single"
scope.glitch.repeat = 1
scope.io.hs2 = "glitch"
scope.glitch.offset = 10
scope.glitch.width = 3
```

Result:

- Faulty Ciphertext: `56 9b 6f 66 c9 41 96 f7 9c f1 49 44 29 13 04 e4`
- Correct Ciphertext: `f5 d3 d5 85 03 b9 69 9d e7 85 89 5a 96 fd ba af`

AES Fault Analysis: Rounds 8–10

After Fault Injection at Start of 8^{th} Round

No Fault

96	$e9$	$e9$	$3c$
71	87	61	89
$6a$	91	04	13
$e4$	$c7$	90	ff

With Fault

96	$e9$	$e9$	$3c$
71	$a3$	61	89
$6a$	91	04	13
$e4$	$c7$	90	ff

Diff

00	00	00	00
00	24	00	00
00	00	00	00
00	00	00	00

After Round 8 (SubBytes, ShiftRows, MixColumns, AddRoundKey)

$0c$	$b7$	$3b$	ad
77	$b8$	$a0$	$c3$
31	$0a$	19	$d8$
43	$b0$	70	eb

$2b$	$b7$	$3b$	ad
$4d$	$b8$	$a0$	$c3$
$2c$	$0a$	19	$d8$
$5e$	$b0$	70	eb

27	00	00	00
3a	00	00	00
1d	00	00	00
1d	00	00	00

AES Fault Analysis: Rounds 8–10

After Round 9

$c2$	10	$a5$	54
df	31	da	$c0$
67	79	42	$5d$
$9b$	74	40	fa

dc	52	13	$6e$
$d0$	73	$1b$	ec
68	bf	35	$4b$
$8a$	$f0$	$f6$	ec

1e	42	b6	3a
0f	42	c1	2c
0f	c6	77	16
11	84	b6	16

After Round 10 (Final Output)

Correct Ciphertext

$f5$	03	$e7$	96
$d3$	$b9$	85	fd
$d5$	69	89	ba
85	$9d$	$5a$	af

Faulty Ciphertext

56	$c9$	$9c$	29
$9b$	41	$f1$	13
$6f$	96	49	04
66	$f7$	44	$e4$

Difference

a3	ca	7b	bf
48	f8	74	ee
ba	ff	c0	be
e3	6a	1e	4b

After 8th Round

f_1			
f_2			
f_3			
f_4			

After 9th Round

$2f_1$	f_4	f_3	$3f_2$
f_1	f_4	$3f_3$	$2f_2$
f_1	$3f_4$	$2f_3$	f_2
$3f_1$	$2f_4$	f_3	f_2

After 10th Round Shift Row

$2f_1$	f_4	f_3	$3f_2$
f_4	$3f_3$	$2f_2$	f_1
$2f_3$	f_2	f_1	$3f_4$
f_2	$3f_1$	$2f_4$	f_3

If we represent the 10^{th} round key as K_{10}, it can be expressed as:

k_{00}	k_{01}	k_{02}	k_{03}
k_{10}	k_{11}	k_{12}	k_{13}
k_{20}	k_{21}	k_{22}	k_{23}
k_{30}	k_{31}	k_{32}	k_{33}

Using Voltage Glitching

Target: AES-128 on CWLite-ARM

- 8th round identified at sample range: **5080–5120**
- Clock glitch injected at: **5100 samples**

Glitch Configuration:

```
scope.glitch.clk_src = "clkgen"
scope.glitch.output = "glitch_only"
scope.glitch.trigger_src = "ext_single"
scope.io.glitch_lp = True
scope.io.glitch_hp = True
scope.glitch.offset = -37.890625
scope.glitch.width  = 37.109375
```

Result:

- Faulty Ciphertext: `5d 37 58 b2 4b b2 0b 7c 9e 67 58 55 39 b0 2d ab`
- Correct Ciphertext: `f5 d3 d5 85 03 b9 69 9d e7 85 89 5a 96 fd ba af`

AES Fault Analysis: Rounds 8–10

After Fault Injection at Start of 8^{th} Round

Without Fault

96	e9	e9	3c
71	87	61	89
6a	91	04	13
e4	c7	90	ff

With Fault

96	e9	e9	52
52	87	61	89
6a	91	04	13
e4	c7	90	ff

Difference

00	00	00	6e
23	00	00	00
00	00	00	00
00	00	00	00

After Completion of 8^{th} RoundSubByte,ShiftRow,MixColumn,AddRoundkey(8)

Without Fault

0c	b7	3b	ad
77	b8	a0	c3
31	0a	19	d8
43	b0	70	eb

With Fault

0c	b7	3b	9e
77	b8	a0	75
31	0a	19	90
43	b0	70	6e

Difference

00	00	00	33
00	00	00	b6
00	00	00	48
00	00	00	85

AES Fault Analysis: Rounds 8–10

After The Completion of 9^{th} Round(SubByte,ShiftRow,MixColumn,AddRoundkey(9))

Without Fault

c2	10	a5	54
df	31	da	c0
67	79	42	5d
9b	74	40	fa

With Fault

b4	11	6b	73
a9	32	a7	5e
fd	7b	f1	c3
77	75	f3	43

Difference

76	01	ce	27
76	03	7d	9e
9a	02	b3	9e
ec	01	b3	b9

After The Completion of 10^{th} Round(SubByte,ShiftRow,AddRoundkey(10)) :

Without Fault

f5	03	e7	96
d3	b9	85	fd
d5	69	89	ba
85	9d	5a	af

With Fault

5d	4b	9e	39
37	b2	67	b0
58	0b	58	2d
b2	7c	55	ab

Difference

a8	48	79	af
e4	0b	e2	4d
8d	62	d1	97
37	e1	0f	04

After 8th Round

			f_1
			f_2
			f_3
			f_4

After 9th Round

f_4	f_3	$3f_2$	$2f_1$
f_4	$3f_3$	$2f_2$	f_1
$3f_4$	$2f_3$	f_2	f_1
$2f_4$	f_3	f_2	$3f_1$

After 10th Round Shift Row

f_4	f_3	$3f_2$	$2f_1$
$3f_3$	$2f_2$	f_1	f_4
f_2	f_1	$3f_4$	$2f_3$
$3f_1$	$2f_4$	f_3	f_2

If we represent the 10^{th} round key as K_{10}, it can be expressed as:

k_{00}	k_{01}	k_{02}	k_{03}
k_{10}	k_{11}	k_{12}	k_{13}
k_{20}	k_{21}	k_{22}	k_{23}
k_{30}	k_{31}	k_{32}	k_{33}

Flipping Bits to Break BipBip

Key Points about BipBip Cipher

- **Tweakable block cipher** tailored for fast decryption in ASICs.
- Uses an unconventional **24-bit block size**, a **256-bit master key**, and a **40-bit tweak**.
- Optimized for latency-sensitive applications (e.g., embedded systems, IoT).
- **Decryption-oriented design** — emphasizes ciphertext-to-plaintext transformation.
- **Decryption Structure Overview:**

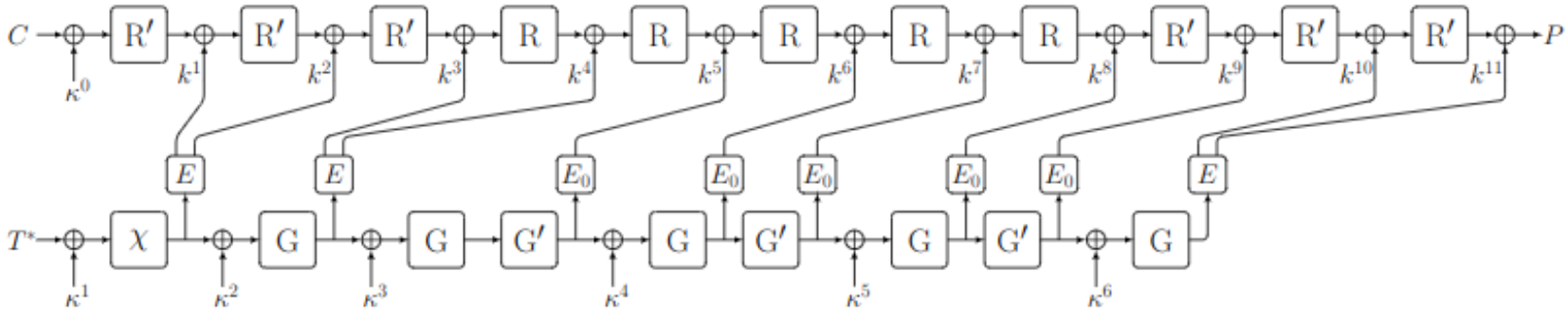


Figure: High Level Decryption Structure of BipBip

Attack Model

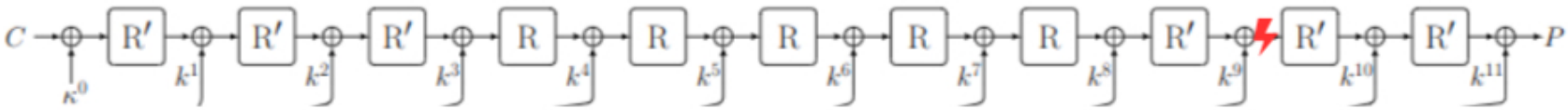


Figure: Proposed fault injection attack

- A single bit fault at the start of 9th round s-box operation is required.
- Need such 4 Faulty values where single bit is flipped in the input of 9th round s-box.

Power Trace of BipBip on CWLite-ARM

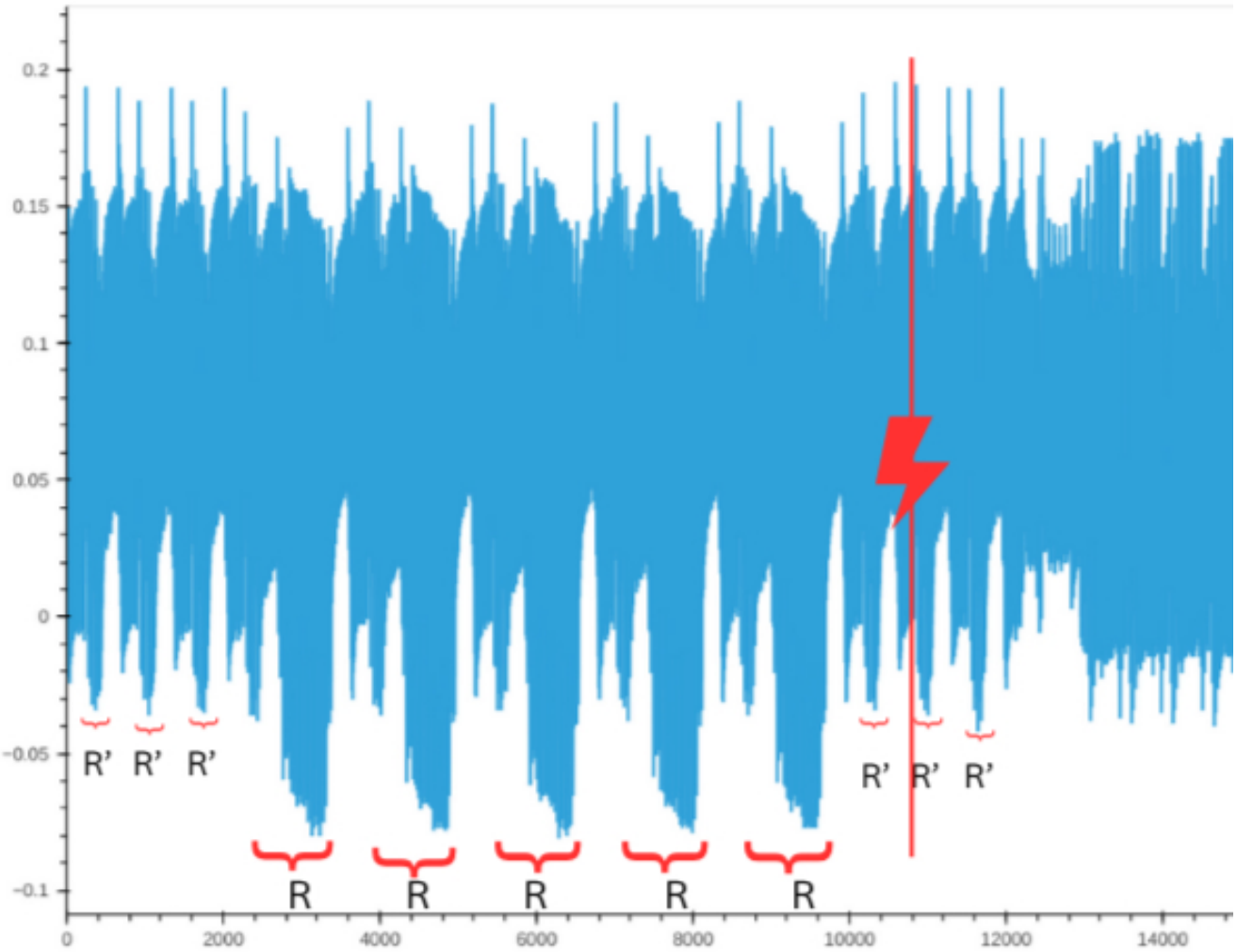


Figure: Power trace BipBip on CWLite-ARM

Glitch Attack Settings on BipBip Implementation

- **Initial Step:** Integrated `sempleserial` for communication between target and scope.
- **Attack Window:** Identified using BipBip power trace on CWLite — approx. between samples `10820 to 10890`.
- **CWLite Settings:**
  - `scope.glitch.clk_src = "clkgen"`
  - `scope.glitch.output = "clock_xor"`
  - `scope.glitch.trigger_src = "ext_single"`
  - `scope.glitch.repeat = 1`
  - `scope.io.hs2 = "glitch"`
- **CWHusky Settings:**
  - `scope.glitch.clk_src = "pll"`
- Attack location on CWHusky determined from BipBip's power trace on that platform.

Analysis of the attack

Table: Summary of Clock Glitch Fault Injection Parameters and Results

Location	Width	Offset	Plaintext	Faulty vs Correct state (Input of Round 9)
10841	1.17	1.17	052aa7	**Faulty:** 011100 100111 111101 101111 **Correct:** 001100 100111 111101 101111
10842	1.95	1.17	6462d1	**Faulty:** 000100 100111 111101 101111 **Correct:** 001100 100111 111101 101111
10855	1.9	1.7	6f2a99	**Faulty:** 001000 100111 111101 101111 **Correct:** 001100 100111 111101 101111
10866	1.17	1.17	3c12bc	**Faulty:** 001101 100111 111101 101111 **Correct:** 001100 100111 111101 101111
10878	1.17	1.17	105bfc	**Faulty:** 001100 100101 111101 101111 **Correct:** 001100 100111 111101 101111
10887	1.9	1.17	1c52e4	**Faulty:** 001100 101111 111101 101111 **Correct:** 001100 100111 111101 101111

Preparing PQC for Fault Analysis: A Kyber Implementation

NIST PQC Standardization Overview

- **Goal:** Identify quantum-resistant public-key cryptographic algorithms for standardization.
- **Initiated:** December 2016 by NIST in response to the threat posed by quantum computers.
- **Process:**
  - Round 1: 69 submissions (Dec 2017)
  - Round 2: 26 candidates (Jan 2019)
  - Round 3: 7 finalists + 8 alternates (July 2020)
  - **Final Selections (July 2022):**
    - **Encryption/KEM:** Kyber (CRYSTALS-Kyber)
    - **Signatures:** Dilithium, Falcon, SPHINCS+
- **Importance:** Ensures secure communication in a post-quantum world, replacing RSA and ECC.

Detailed Comparison of Kyber Variants

Variant	Security Level (bits)	Public Key (bytes)	Private Key (bytes)	Ciphertext (bytes)
Kyber512	128	800	1,632	768
Kyber768	192	1,184	2,400	1,088
Kyber1024	256	1,568	3,168	1,568

Future Work