Fault Attacks on Block Ciphers
(Attacks and Automated Evaluation)

Chester Rebeiro

IIT Madras

Modern ciphers
designed with very strong assumptions

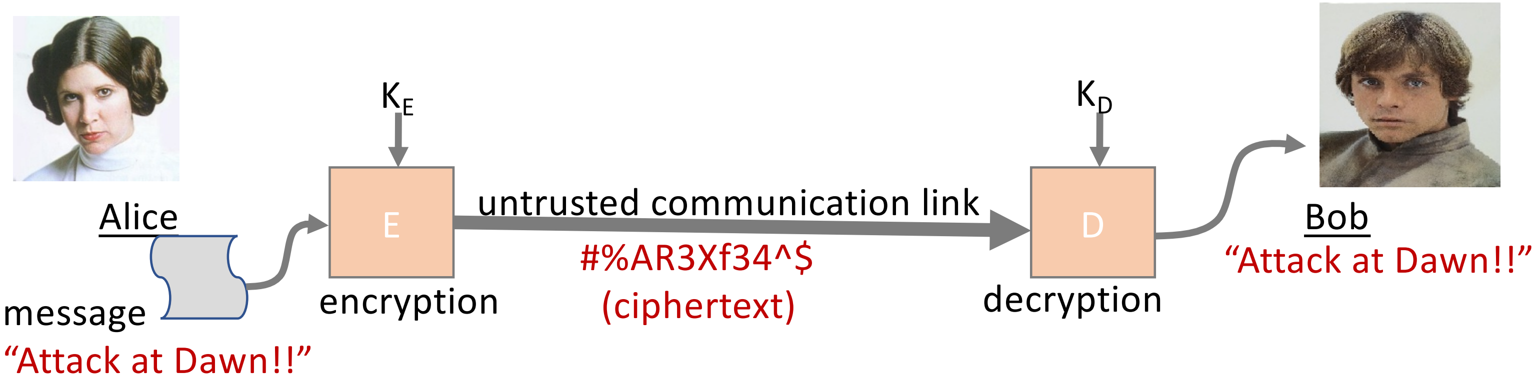
- **Kerckhoff's Principle**
  - The system is completely known to the attacker. This includes encryption & decryption algorithms, plaintext
  - only the key is secret

- Why do we make this assumption?
  - Algorithms can be leaked (secrets never remain secret)
  - or reverse engineered

Mallory's task is therefore very difficult….

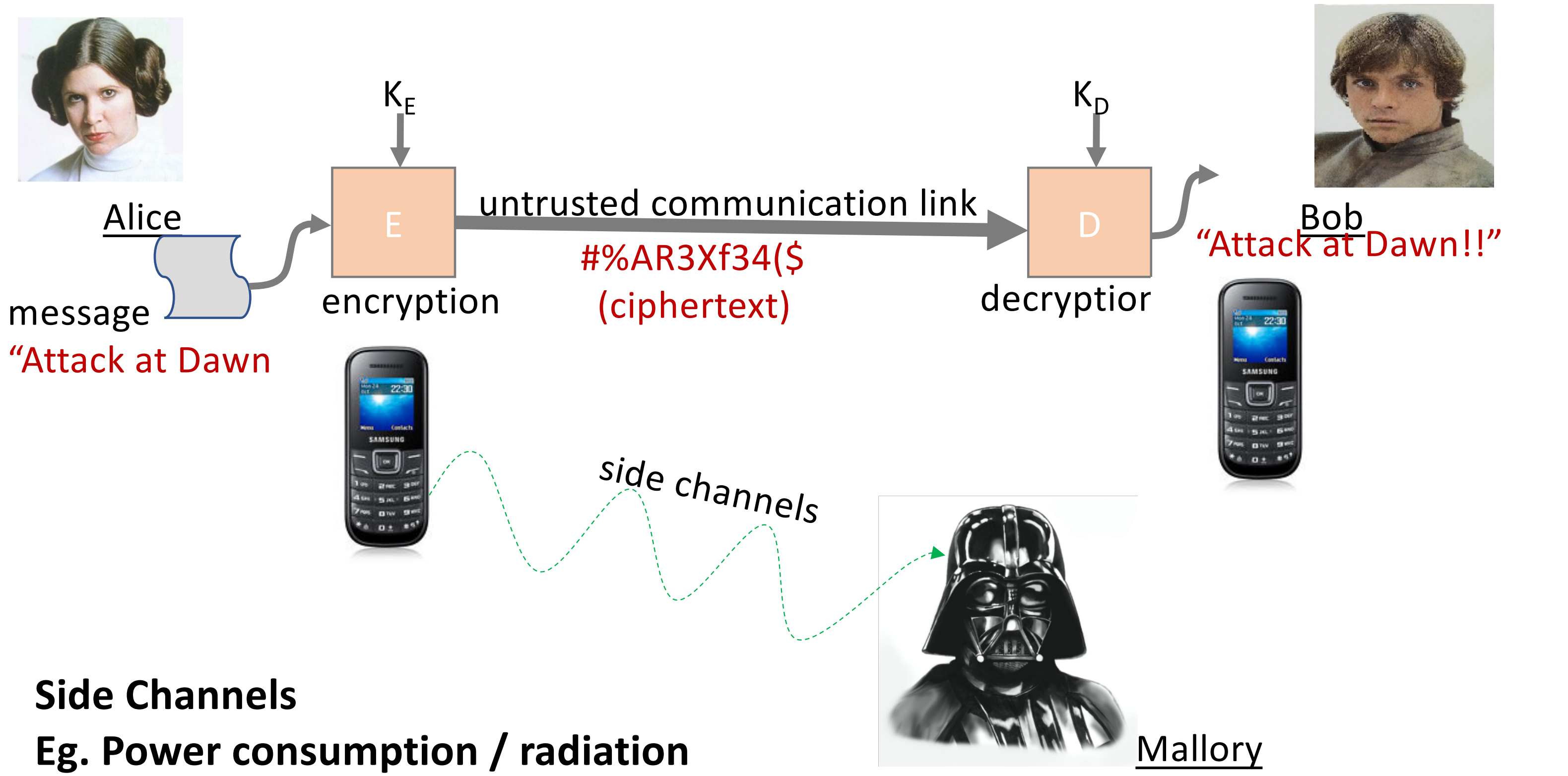
Security as strong as its weakest link

- Mallory just needs to find the weakest link in the system ….there is still hope!!!



Side Channels

Side Channel Analysis
(the weak links)

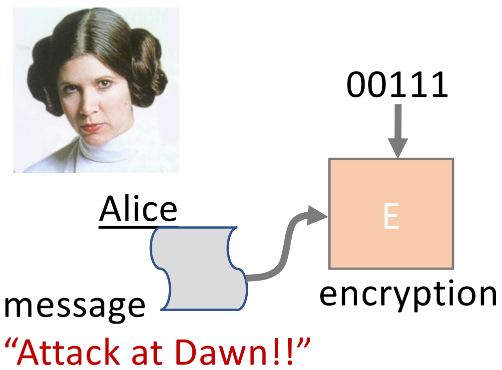


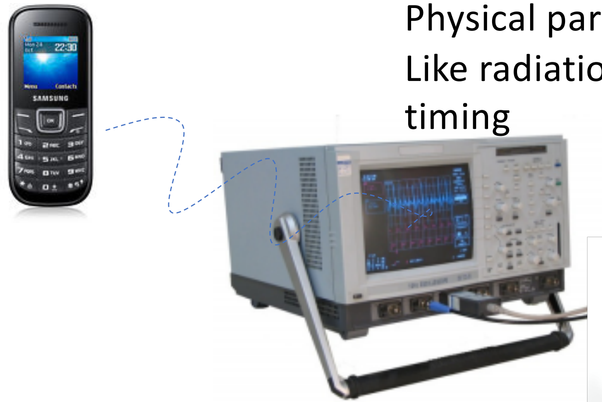
Side Channels
Eg. Power consumption / radiation of device, execution time, etc.

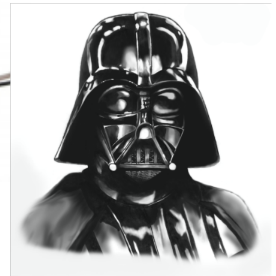
Gets information about the keys by monitoring Side channels of the device

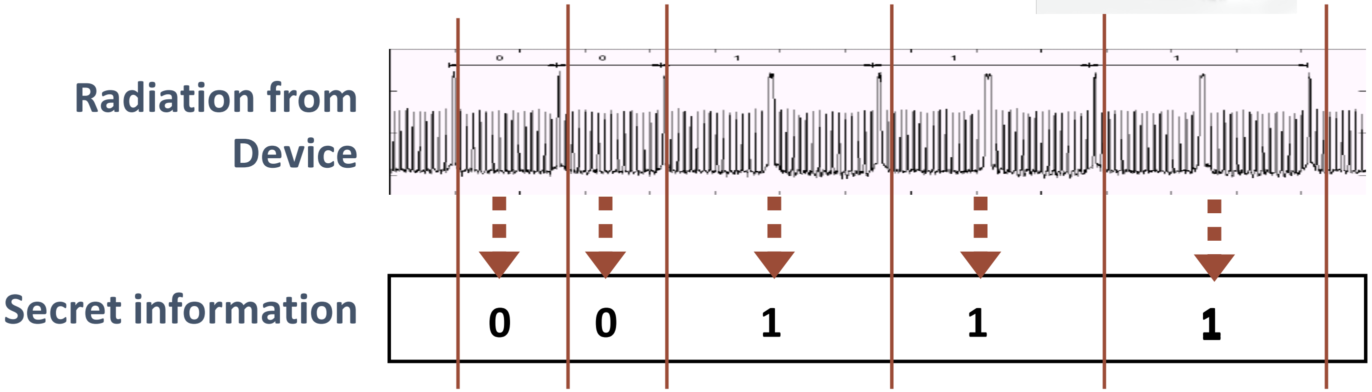
Side Channel Analysis

Mallory measures some
Physical parameter of the device
Like radiation, power consumption or
timing





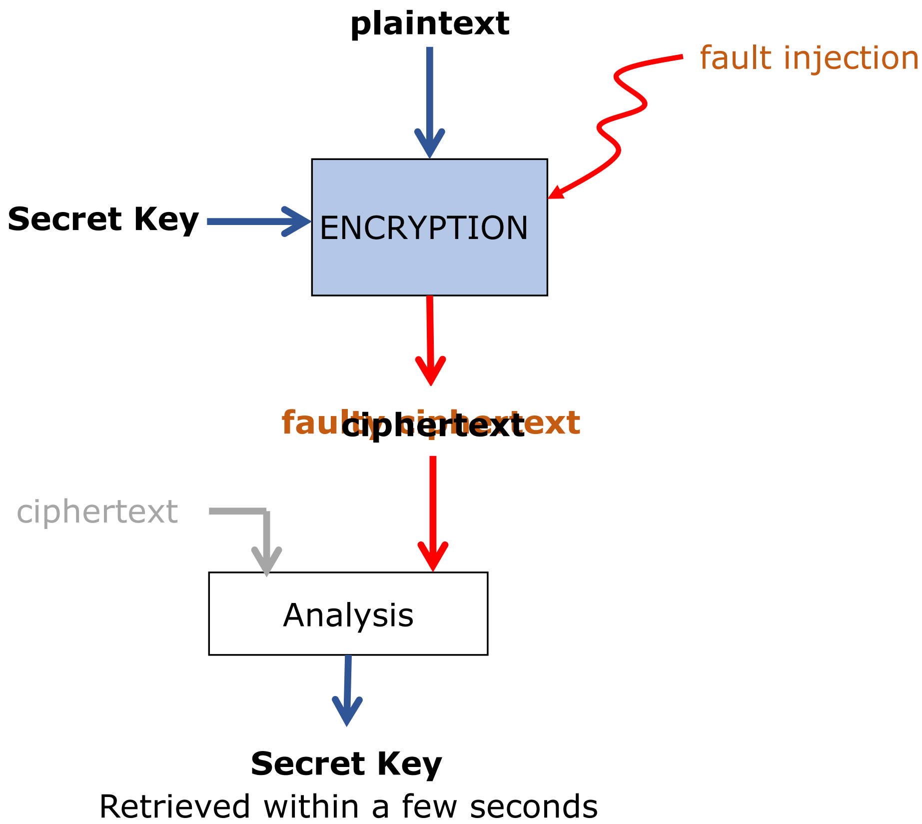




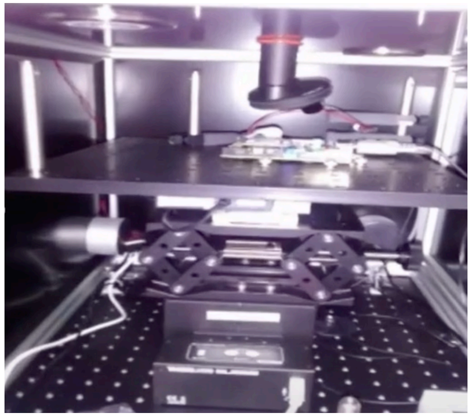
Types of Side Channel Attacks

	Passive Attacks	Active Attack
Non-Invasive Attacks	Side-channel attacks: timing attack, Power+EM attacks, cache trace	Insert fault in device without depackaging: using clock glitch, power glitch, change in temperature
Semi—invasive attacks (device is depackaged but no direct electrical contact is made to the chip surface)	Read out memory of device without probing or using the normal read out circuits	Induce faults in depackaged devices with x-rays, EM fields, or optical mechanisms
Invasive Attacks (no limits on what is done with the device)	Probing depackaged device and observe data signals	Depackaged devices are manipulated by probing using laser beams, and focused ion beams.

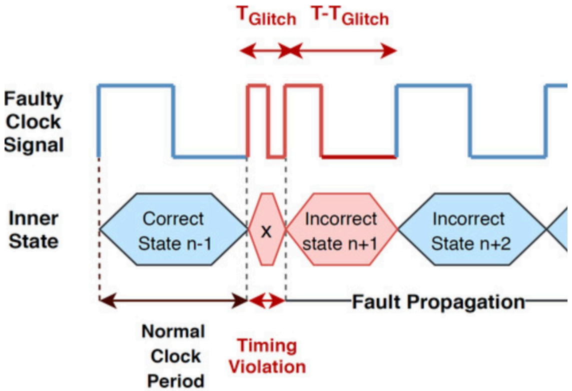
Fault Attacks on Ciphers



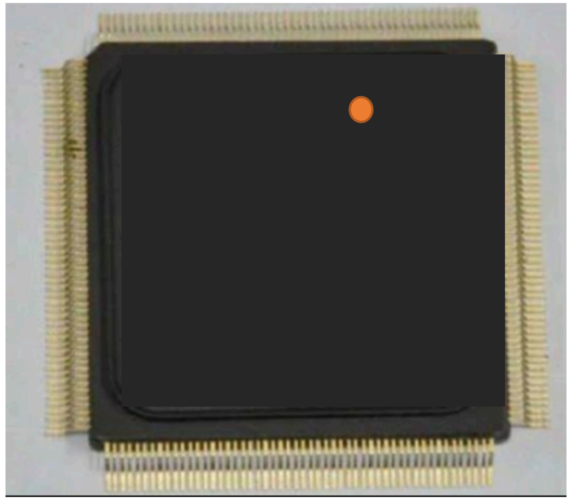
Fault injection needs to be precise



Laser fault injection



Clock glitch fault injection



Attributes of a fault:

-- X, Y coordinates for fault injection

-- time instant laser to be turned on

-- laser intensity

-- type of fault (random / stuck at)

Exploitable Fault depend considerably on the cipher algorithm

Fault Models

- Bit model : When fault is injected, exactly one bit in the state is altered

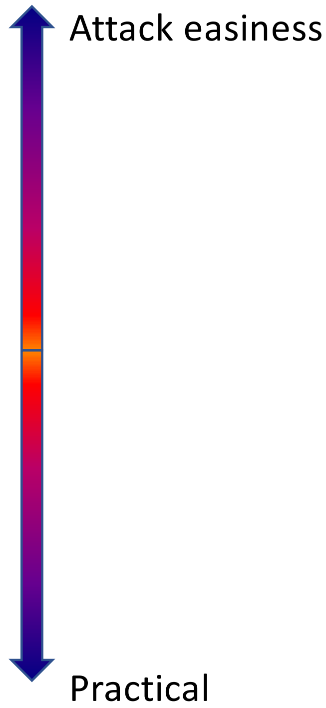
  eg. 8823124345 → 8833124345

- Byte model : exactly one byte in the state is altered

  eg. 8823124345 → 8836124345

- Multiple byte model : faults affect more than one byte

  eg. 8823124345 → 8836124333



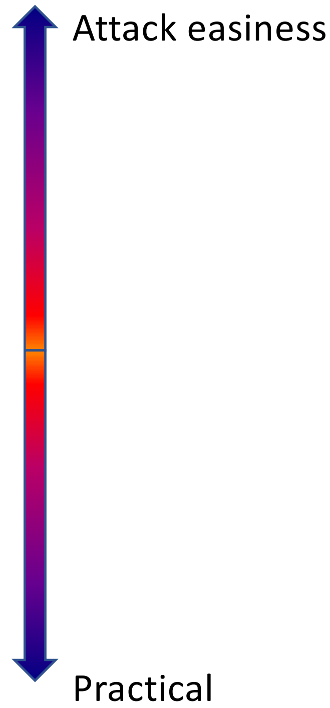
Fault Models

- Stuck at Fault : When fault is injected, bits forced to be stuck at 0 (or 1)

  eg. 882312434F → 8823124340

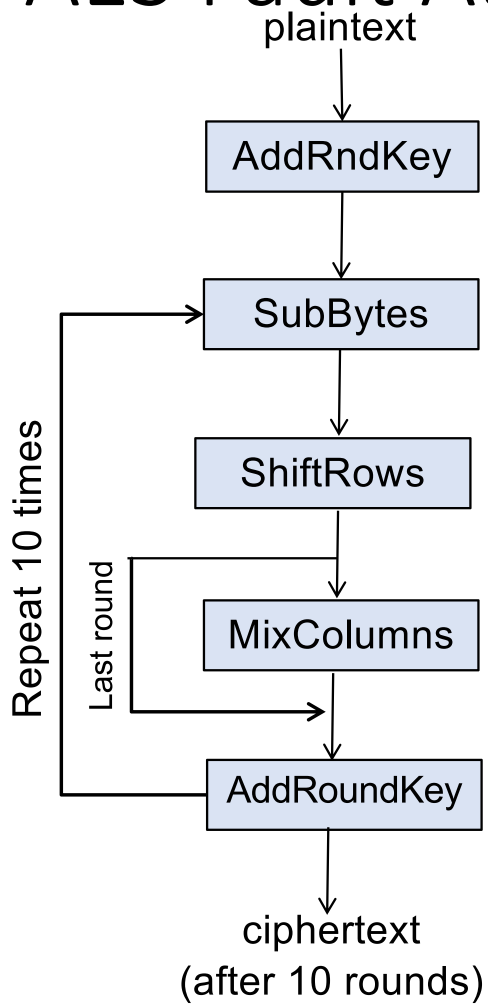
- Transient random model : data is randomly altered for a short duration

  eg. 8823124345 → 8836124345



Fault injection is difficult…. The attacker would want to reduce the number of faults to be injected

AES Fault Attacks – A Case Study



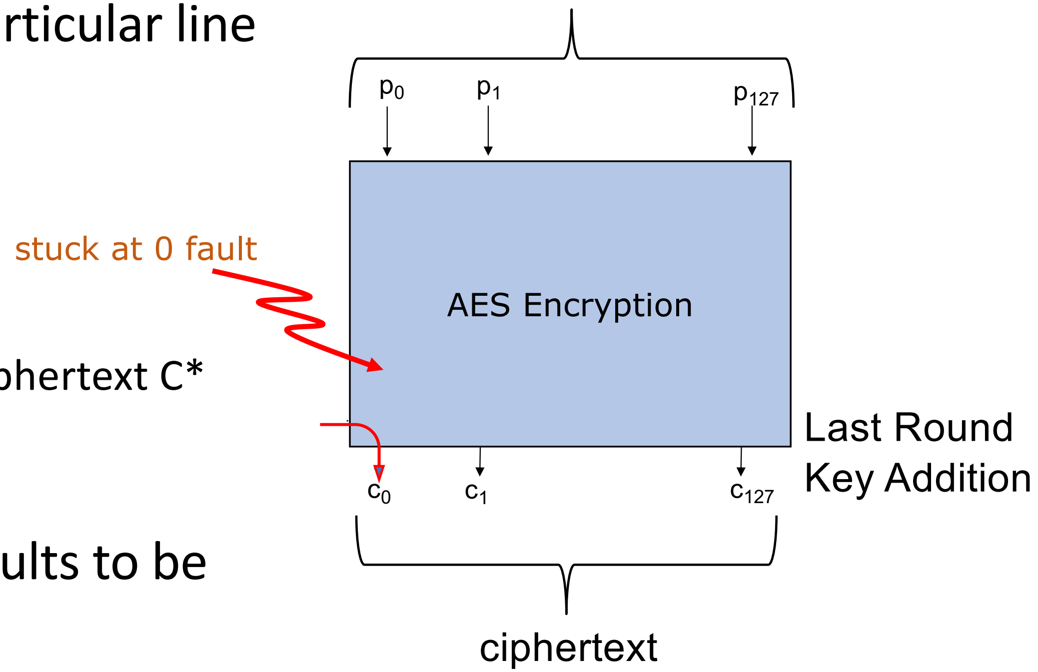
A Simple Fault Attack on AES

- Attacker has the capability of resetting a particular line during the AES round key addition.

  (i.e. exactly one bit is reset)

- Attack Procedure
  1. Encrypt plaintext to get ciphertext C
  2. Inject fault in the i-th bit as shown. Get the ciphertext C*
  3. If C=C*, we infer $K_i = 1$
     If C≠C*, we infer $K_i = 0$

- This techniques requires 128 stuck at 0 faults to be injected.
  - difficult… can we do better?



Differential Properties of a cipher

Fault attackers look to solve equations of the form:

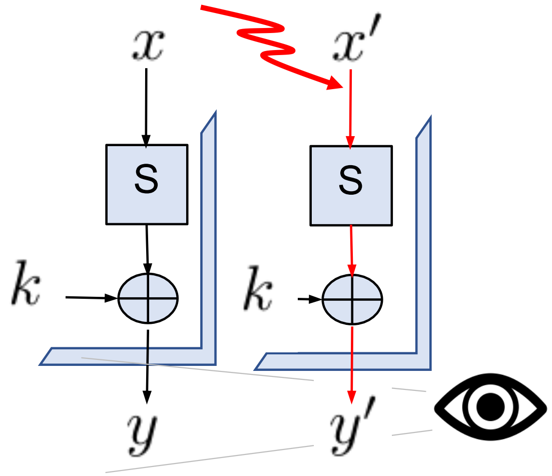
$S^{-1}(y \oplus k) \oplus S^{-1}(y' \oplus k) = \delta$

where y and y' are known

If δ is known, number of solutions for k is very small

but, we do not know δ, so try every possible value of δ.

$$\delta = x \oplus x'$$



Improving the AES attack last round, bit fault

$$S^{-1}(y \oplus k) \oplus S^{-1}(y' \oplus k) = \delta$$

Suppose δ is a bit fault

$\delta = \{(00000001), (00000010), (00000100), (0001000), \cdots, (10000000)\}$

Each value of δ will give one solution for k.
Thus, 8 solutions for k

x

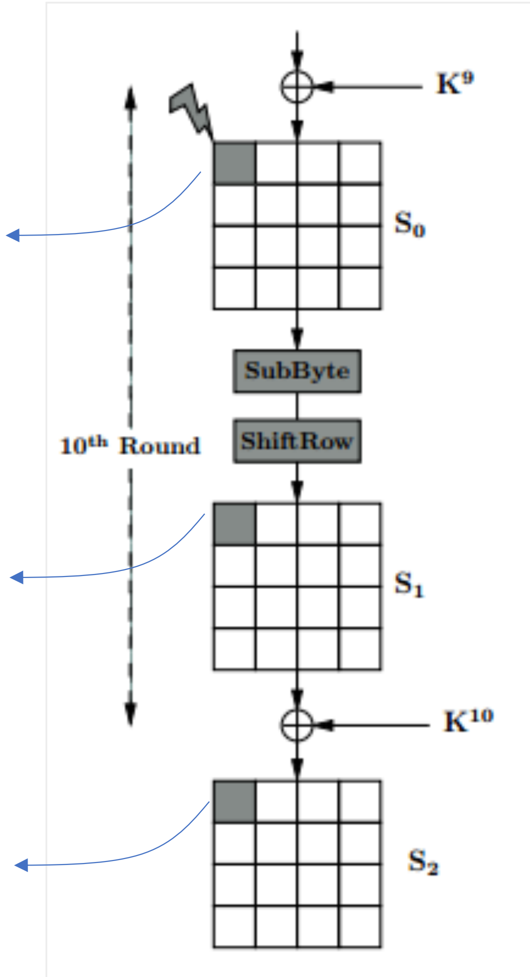
x'

$S(x)$

$S(x')$

$y = S(x) \oplus k$

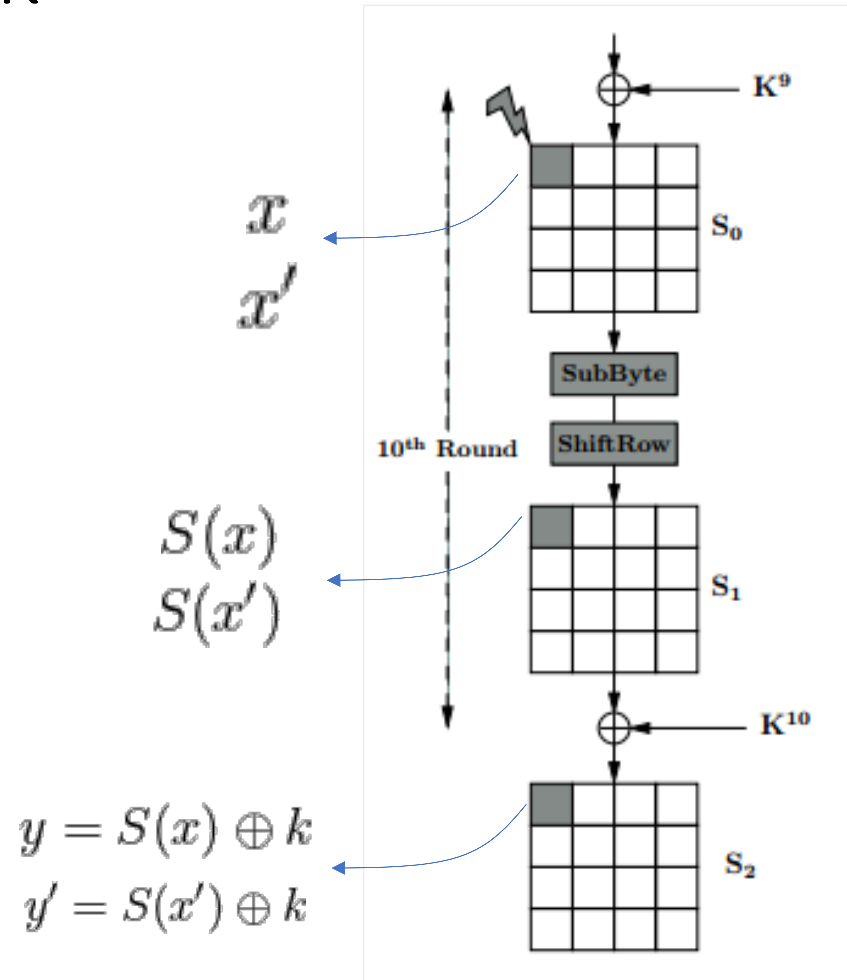
$y' = S(x') \oplus k$



Improving the AES attack last round, bit fault

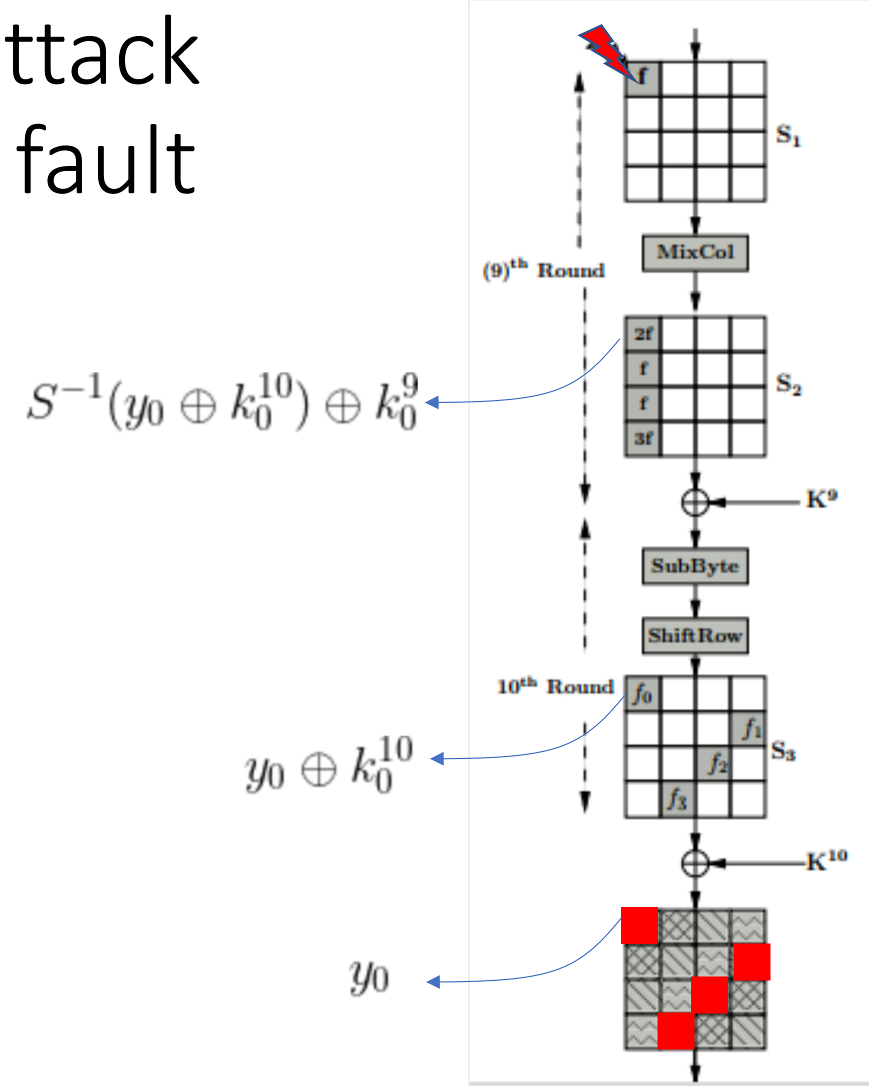
16 keys, prese δ Thus total complexity of the attack is 8^{16} (approximately 2^{48}).

16 bit faults required.



Improving the AES attack
9-th round, random fault

$$S^{-1}(y_0 \oplus k_0^{10}) \oplus S^{-1}(y_0' \oplus k_0^{10}) = 2\delta$$
$$S^{-1}(y_7 \oplus k_7^{10}) \oplus S^{-1}(y_7' \oplus k_7^{10}) = \delta$$
$$S^{-1}(y_{10} \oplus k_{10}^{10}) \oplus S^{-1}(y_{10}' \oplus k_{10}^{10}) = \delta$$
$$S^{-1}(y_{13} \oplus k_{10}^{13}) \oplus S^{-1}(y_{13}' \oplus k_{10}^{13}) = 3\delta$$



Complexity to δ ve the 4 equations is 2^8.
Will deliver 4 key bytes (32 bits).

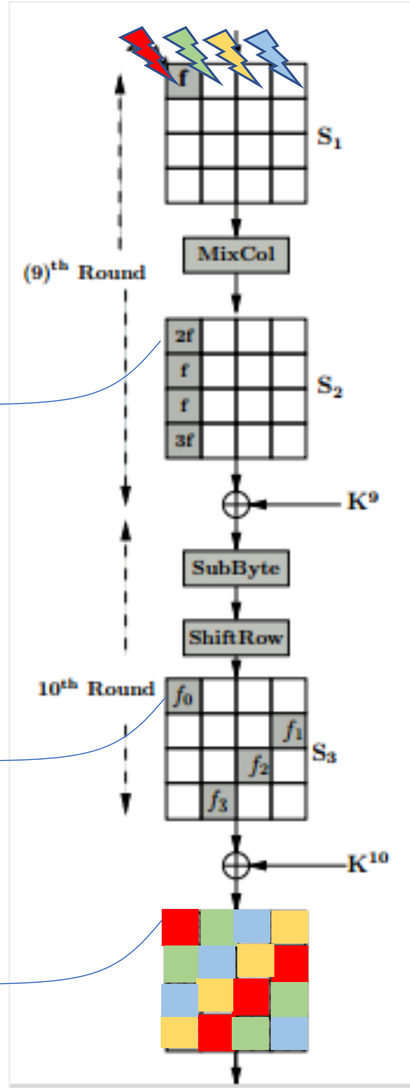
Improving the AES attack
9-th round, random fault

$$S^{-1}(y_0 \oplus k_0^{10}) \oplus S^{-1}(y'_0 \oplus k_0^{10}) = 2\delta$$
$$S^{-1}(y_7 \oplus k_7^{10}) \oplus S^{-1}(y'_7 \oplus k_7^{10}) = \delta$$
$$S^{-1}(y_{10} \oplus k_{10}^{10}) \oplus S^{-1}(y'_{10} \oplus k_{10}^{10}) = \delta$$
$$S^{-1}(y_{13} \oplus k_{10}^{13}) \oplus S^{-1}(y'_{13} \oplus k_{10}^{13}) = 3\delta$$

$$S^{-1}(y_0 \oplus k_0^{10}) \oplus k_0^9$$

$$y_0 \oplus k_0^{10}$$

$$y_0$$

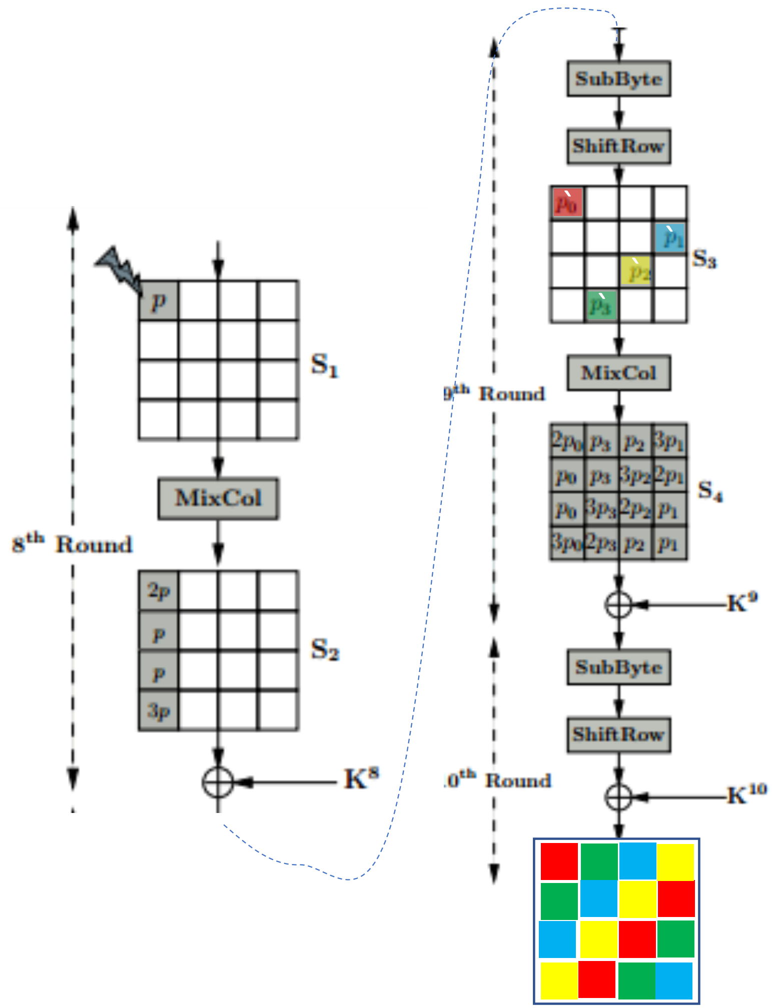


Complexity to δ ve the 4 equations is 2^8.
Will deliver 4 key bytes (32 bits).

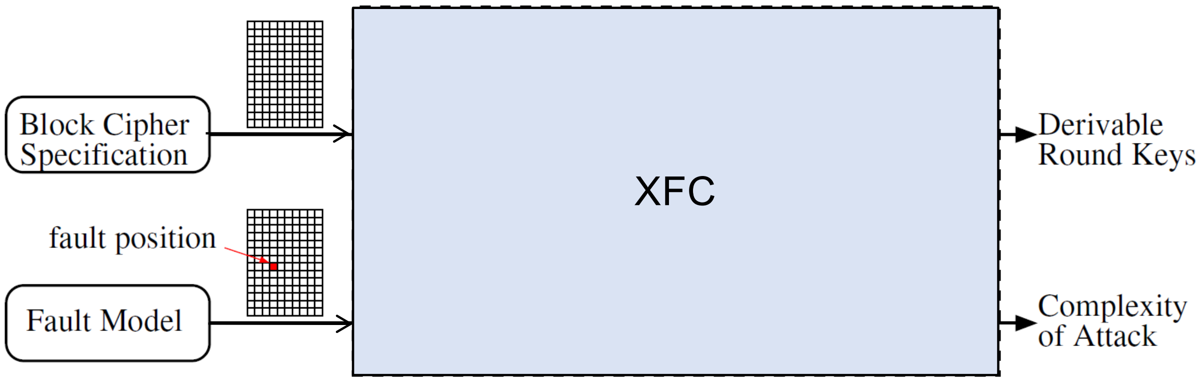
4 faults required to get 128 bits of key.
Total complexity, $2^{8*4}=2^{32}$

Improving the AES attack 8th round, random fault

- A single fault injected in the 8th round will spread to 4 bytes in the 9th round.
- This is equivalent to having 4 faults in each of the 4 columns.
- A single fault can thus be used to determine all key bytes.
- The offline key space is 2^{32} as before. This can be reduced to 2^{8} using the key expansion algorithm



XFC: A Framework for Exploitable Fault Characterization in Block Ciphers



Cipher	$\mathcal{F}_i$	Round Number	#Derived Keys	Offline Complexity
AES	1-27	1-7	0	N/A
	28-31	**7-8**	**128**	$\mathbf{2^8}$
	32-35	8-9	32	2^8
	36-40	9-10	0	N/A

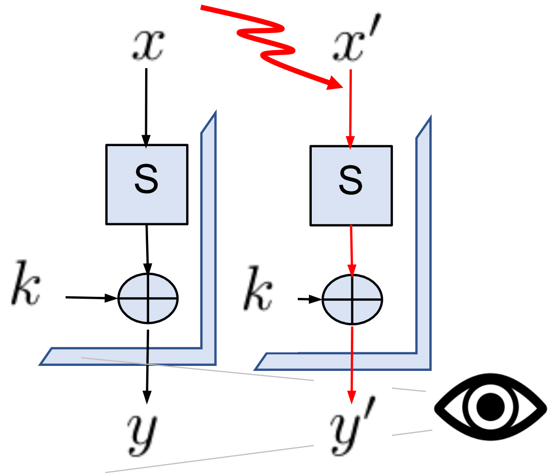
The Central Idea

Fault attackers look to solve equations of the form:

$S^{-1}(y \oplus k) \oplus S^{-1}(y' \oplus k) = \delta$

where y and y' are known, k and δ are unknown.

$\delta = x \oplus x'$



The Central Idea

If multiple equations of this form are found, then the complexity is reduced considerably

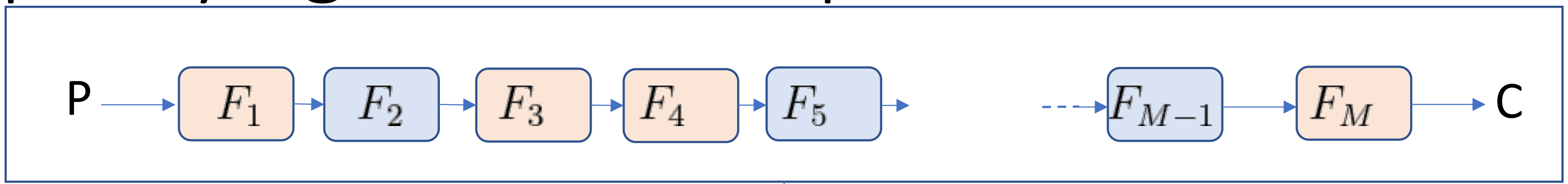
$$
\begin{aligned}
S^{-1}(y_1 \oplus k_1) \oplus S^{-1}(y'_1 \oplus k_1) &= g_1(\delta) \\
S^{-1}(y_2 \oplus k_2) \oplus S^{-1}(y'_2 \oplus k_2) &= g_2(\delta) \\
S^{-1}(y_3 \oplus k_3) \oplus S^{-1}(y'_3 \oplus k_3) &= g_3(\delta) \\
\vdots \qquad \vdots \qquad \vdots \qquad &\vdots \\
S^{-1}(y_N \oplus k_N) \oplus S^{-1}(y'_N \oplus k_N) &= g_N(\delta)
\end{aligned}
$$

Linear function δ

Only key tuples $(k_1, k_2, \cdots, k_N)$ that satisfy all N equations are potential candidates.

Assuming δ is a byte, we can recover N bytes of key with a complexity of 2^8

Specifying the Block Cipher



$F_i \quad (1 \leq i \leq M)$ are Boolean functions

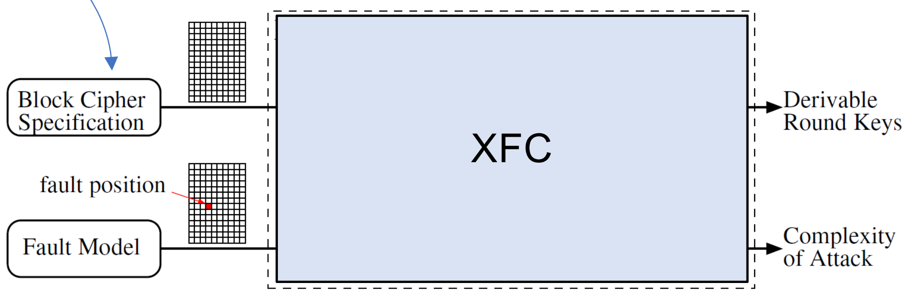
Could be Linear functions

$$F_i(x_1, x_2, \cdots, x_l) = \bigoplus_{j=1}^{l} a_j \cdot x_j$$

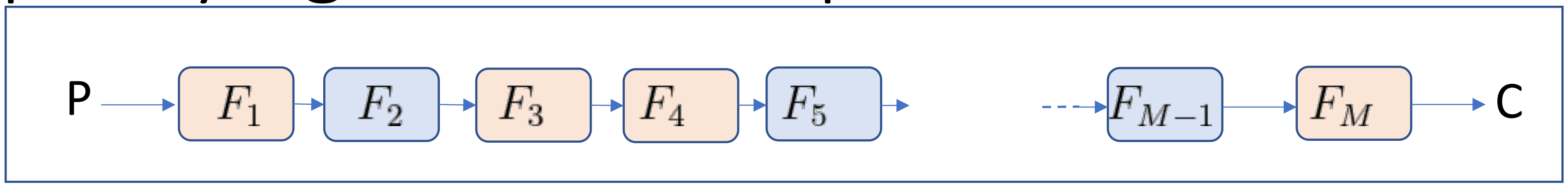
or Non Linear functions

$$F_i(x_1, x_2, \cdots, x_l) = \bigoplus_{j=1}^{l} a_j \prod_{z \in \mathbb{Z}} x_z$$

$a_j \in \{0, 1\}$



Specifying the Block Cipher



$F_i \quad (1 \leq i \leq M)$ are Boolean functions

Could be Linear functions

$$F_i(x_1, x_2, \cdots, x_l) = \bigoplus_{j=1}^{l} a_j \cdot x_j$$

or Non Linear functions

$$F_i(x_1, x_2, \cdots, x_l) = \bigoplus_{j=1}^{l} a_j \prod_{z \in \mathbb{Z}} x_z$$

$a_j \in \{0, 1\}$

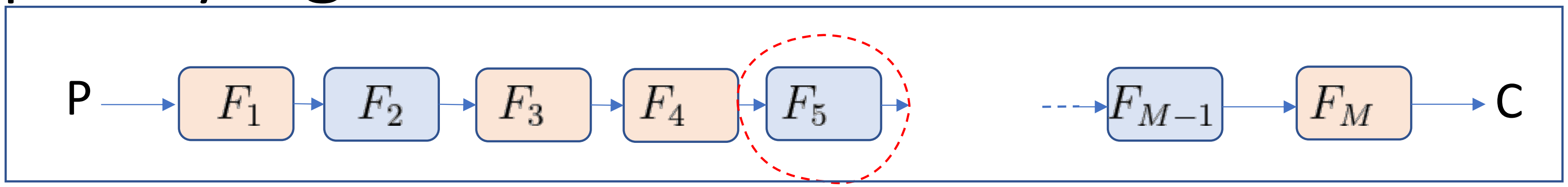
AES Specification

40 functions
F_1 (AddRoundKey) is **Linear**
F_2 (SubBytes of Round 1) is **Non-Linear**
F_3 (ShiftRows of Round 1) is **Linear**
F_4 (MixColumns of Round 1) is **Linear**

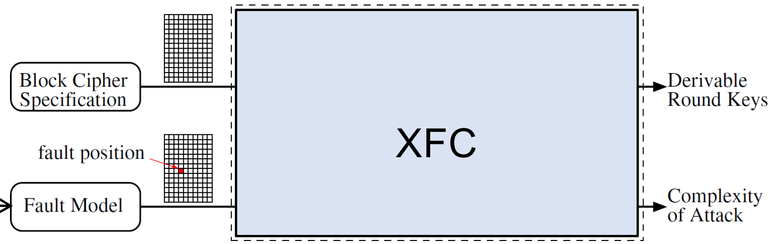
F_5 (AddRoundKey) is Linear
F_6 (SubBytes of Round 2) is **Non-Linear**
F_7 (ShiftRows of Round 2) is **Linear**
F_8 (MixColumns of Round 2) is **Linear**

F_{37} (AddRoundKey of Round 9) is **Linear**
F_{38} (SubBytes of Round 10) is **Non-Linear**
F_{39} (ShiftRows of Round 10) is **Linear**
F_{40} (AddRoundKey of Round 10) is **Linear**

Specifying the Fault Location



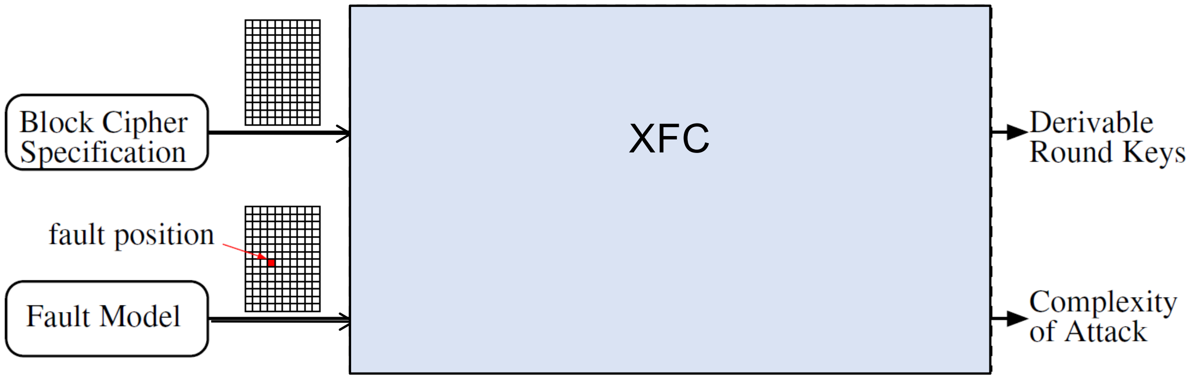
$$f = \langle F_5, 1 \rangle$$



Example:

A Fault in the 1st Byte of the 5th Round SubBytes operation

XFC's Two Phases



(Phase 1) Fault Propagation

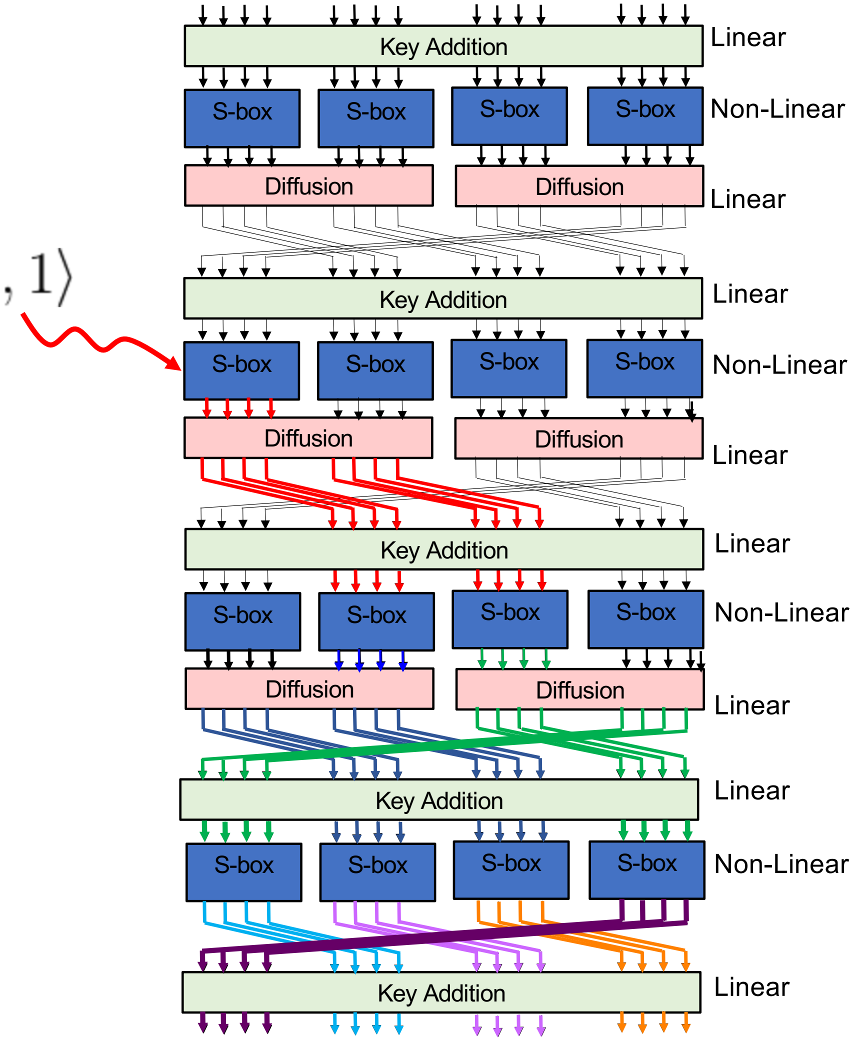
Color the fault affected part.

Propagate and color as follows.

1. When passing through a linear layer, retain same color
2. When passing through a non-linear layer, change color
3. If two bytes of different colors are combined, change the color.

Same colors are linearly correlated
Different colors are not correlated

$$f = \langle F_5, 1 \rangle$$



(Phase 2) Key Determination

Back propagate and try to match colors whenever we hit an s-box

$f = \langle F_5, 1 \rangle$

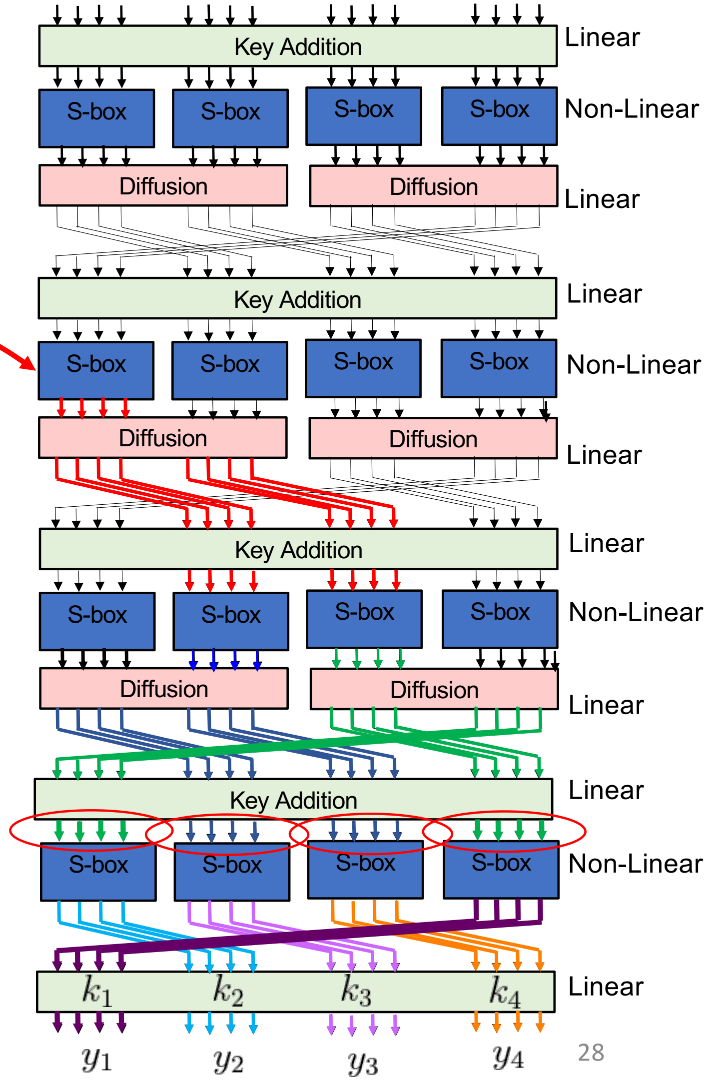
$$S^{-1}(y_1 \oplus k_1) \oplus S^{-1}(y_1' \oplus k_1) = g_1(\delta)$$
$$S^{-1}(y_2 \oplus k_2) \oplus S^{-1}(y_2' \oplus k_2) = g_2(\delta)$$

The offline complexity to find (k_1, k_2) is 2^4 ; ie the possible values δ can take.

Similarly,

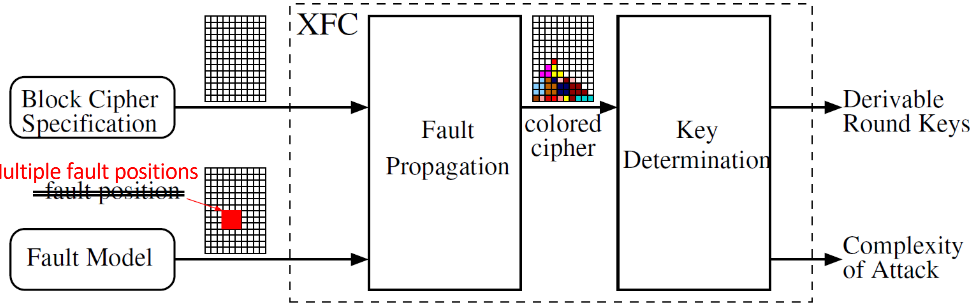
$$S^{-1}(y_3 \oplus k_3) \oplus S^{-1}(y_3' \oplus k_3) = g_3(\delta)$$
$$S^{-1}(y_4 \oplus k_4) \oplus S^{-1}(y_4' \oplus k_4) = g_4(\delta)$$

Can be used to determine (k_3, k_4)



Finding the Sweet Spot

Keep the block cipher specification fixed and iterate through all possible fault locations



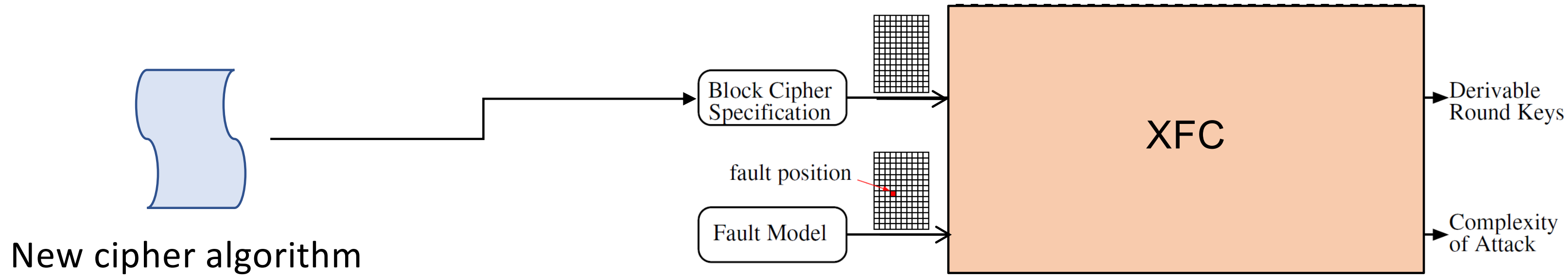
Cipher	$\mathcal{F}_i$	Round Number	#Derived Keys	Offline Complexity
AES	1-27	1-7	0	N/A
	28-31	**7-8**	**128**	$\mathbf{2^8}$
	32-35	8-9	32	2^8
	36-40	9-10	0	N/A
CLEFIA	1-51	1-13	0	N/A
	52-55	13-14	32	2^8
	56-67	**14-17**	**32**	$\mathbf{2^{4.76}}$
	68-74	17-18	0	N/A
SMS4	1-106	1-27	0	N/A
	107-110	**27-28**	**128**	$\mathbf{2^{11.051}}$
	111-114	28-29	96	$2^{3.051}$
	115-118	29-30	64	$2^{3.051}$
	119-122	30-31	32	$2^{3.051}$
	123-128	31-32	0	N/A

Sweet spots

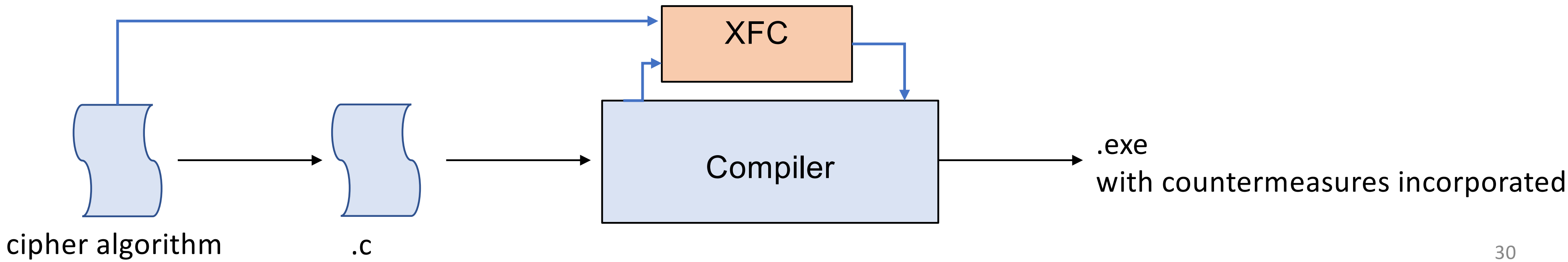
XFC's results are both sound and complete.

Applications of XFC

Automatically evaluate new cipher algorithms for Fault Attacks

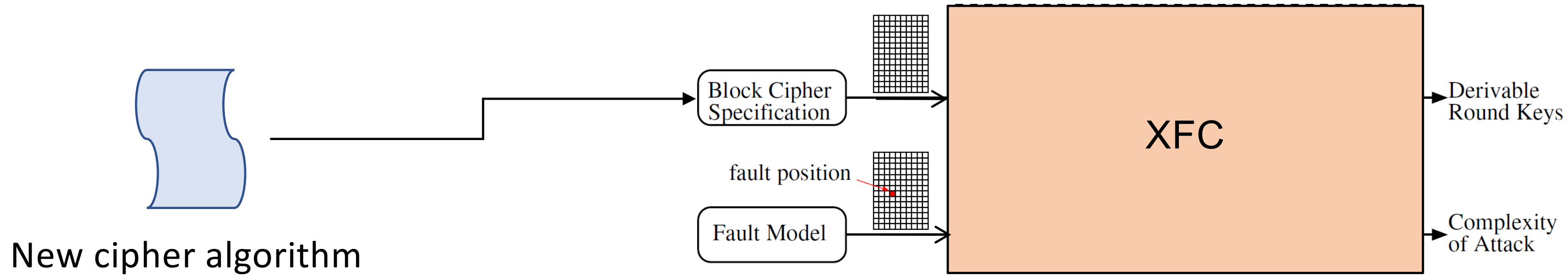


Fault Attack aware Compilers (for software)

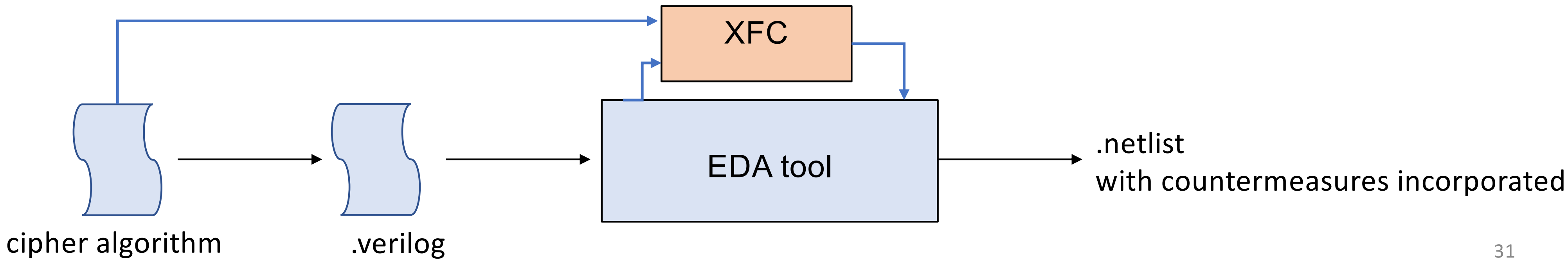


Applications of XFC

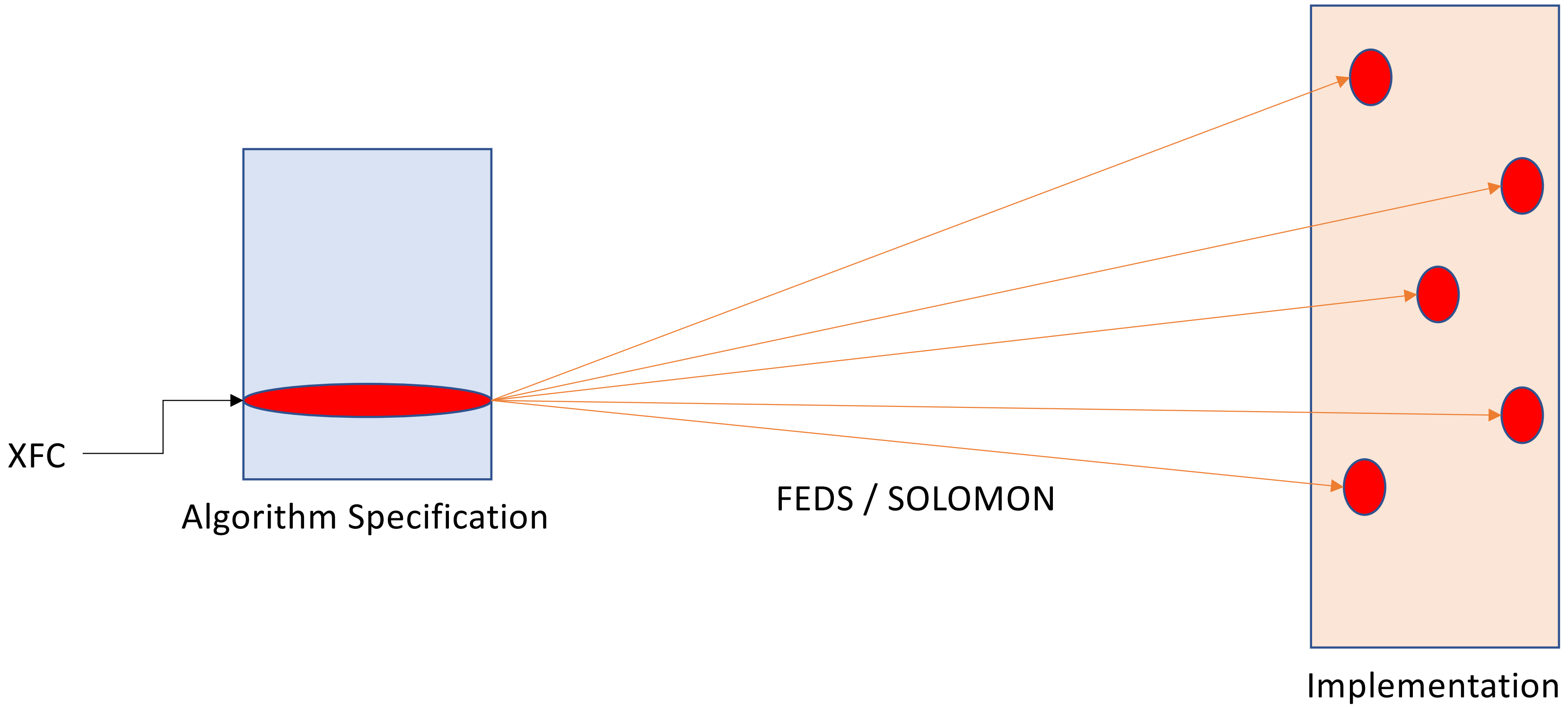
Automatically evaluate new cipher algorithms for Fault Attacks



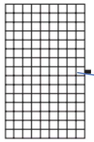
Fault Attack aware EDA tools (for VLSI design)



Mapping fault vulnerable operations to an implementation

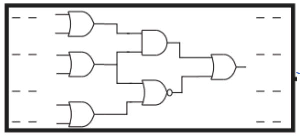


SOLOMON: An automated framework for detecting fault attack vulnerabilities in hardware



BCSL Specification

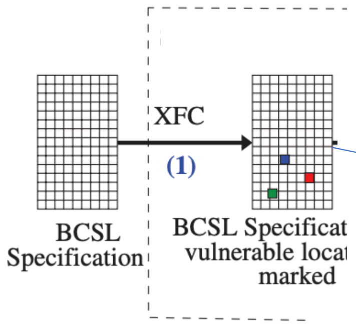
```
⟨ outs ⟩ ⟨ linear ⟩ ⟨ ADDKEY ⟩⟨
        ⟨ outs [1]: {ins[1]}: F_XOR(ins[1], F_LKUP(1, ks))} ⟩
        ⟨ outs [2]: {ins[2]}: F_XOR(ins[2], F_LKUP(2, ks))} ⟩
        ⋮
        ⟨ outs [16]: {ins[16]}: F_XOR(ins[16], F_LKUP(16, ks))} ⟩
/⟩
```



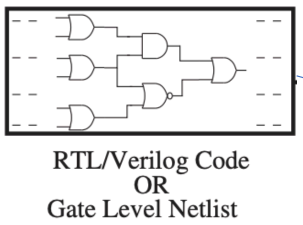
RTL/Verilog Code
OR
Gate Level Netlist

```
module AddRoundKey(in_i, k_i, out_i);
  input [127:0] in_i, k_i;
  output [127:0] out_i;

  assign out_i[127:64] = ((~in_i[127:64]&k_i[127:64])
        |(in_i[127:64]&(~k_i[127:64]));
  assign out_i[63:0] = ((~in_i[63:0]&k_i[63:0])|(in_i
        [63:0]&(~k_i[63:0]));
endmodule
```

SOLOMON: An automated framework for detecting fault attack vulnerabilities in hardware

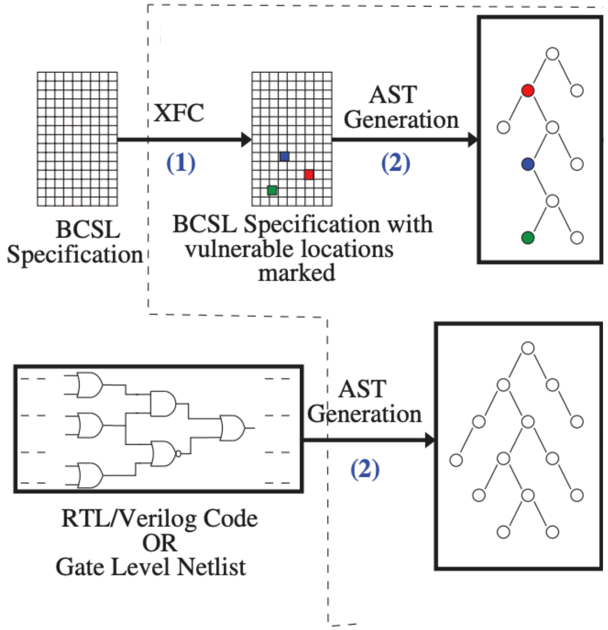


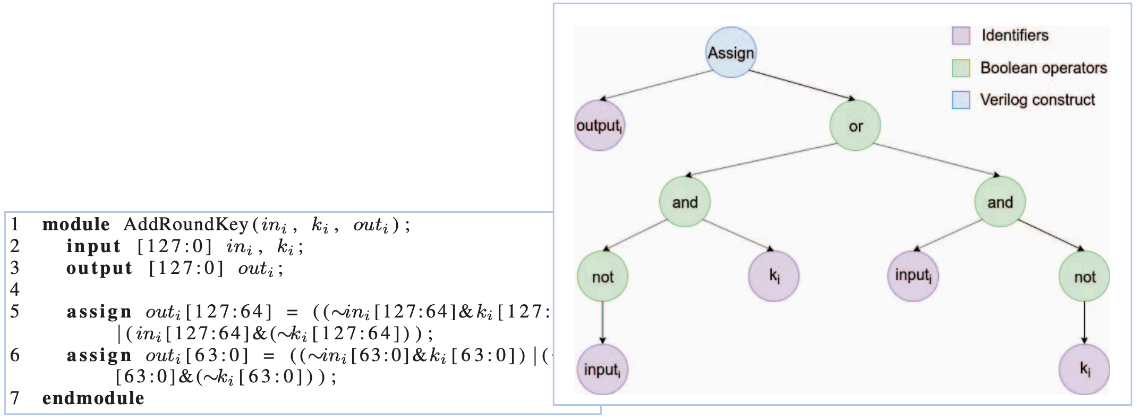

```
⟨ outs ⟩ ⟨ linear ⟩ ⟨ ADDKEY ⟩⟨
        ⟨ outs [1]: {ins[1]}: F_XOR(ins[1], F_LKUP(1, ks))} ⟩
        ⟨ outs [2]: {ins[2]}: F_XOR(ins[2], F_LKUP(2, ks))} ⟩
                :
        ⟨ outs [16]: {ins[16]}: F_XOR(ins[16], F_LKUP(16, ks))} ⟩
/⟩
```



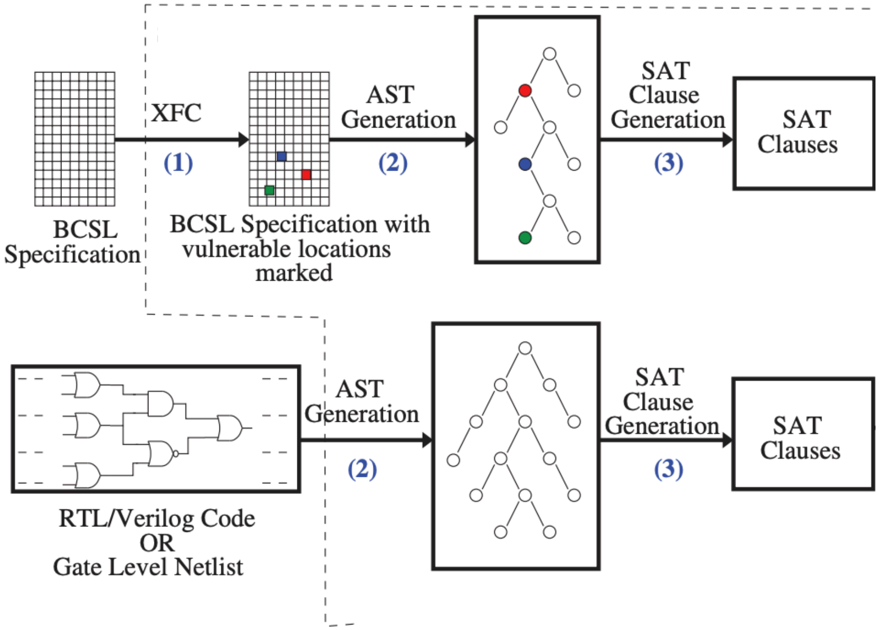

```
module AddRoundKey(in_i, k_i, out_i);
  input [127:0] in_i, k_i;
  output [127:0] out_i;

  assign out_i[127:64] = ((~in_i[127:64]&k_i[127:64])
        |(in_i[127:64]&(~k_i[127:64]));
  assign out_i[63:0] = ((~in_i[63:0]&k_i[63:0])|(in_i
        [63:0]&(~k_i[63:0]));
endmodule
```

SOLOMON: An automated framework for detecting fault attack vulnerabilities in hardware

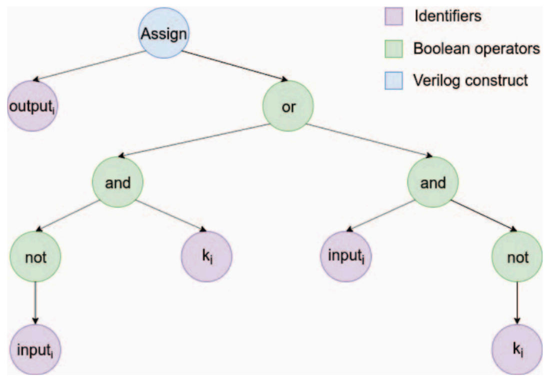




SOLOMON: An automated framework for detecting fault attack vulnerabilities in hardware




```
module AddRoundKey(in_i, k_i, out_i);
  input [127:0] in_i, k_i;
  output [127:0] out_i;

  assign out_i[127:64] = ((~in_i[127:64]&k_i[127:64])
         |(in_i[127:64]&(~k_i[127:64]));
  assign out_i[63:0] = ((~in_i[63:0]&k_i[63:0])|(in_i
         [63:0]&(~k_i[63:0]));
endmodule
```

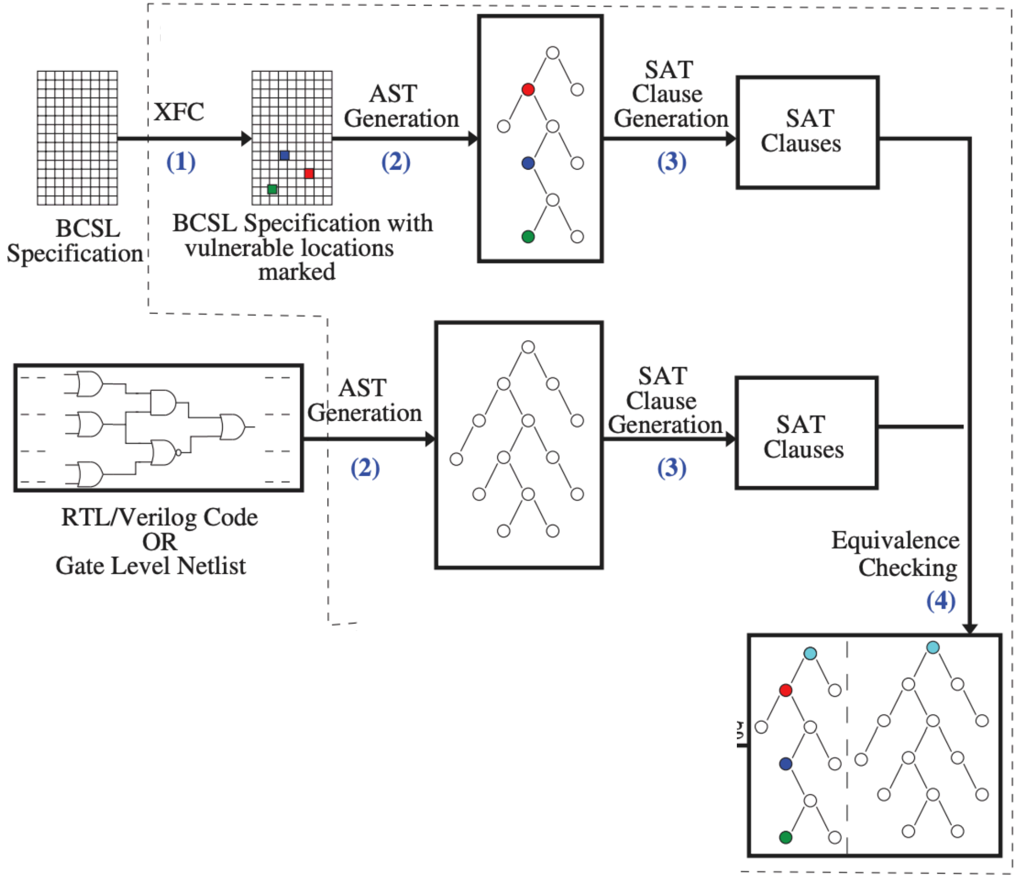


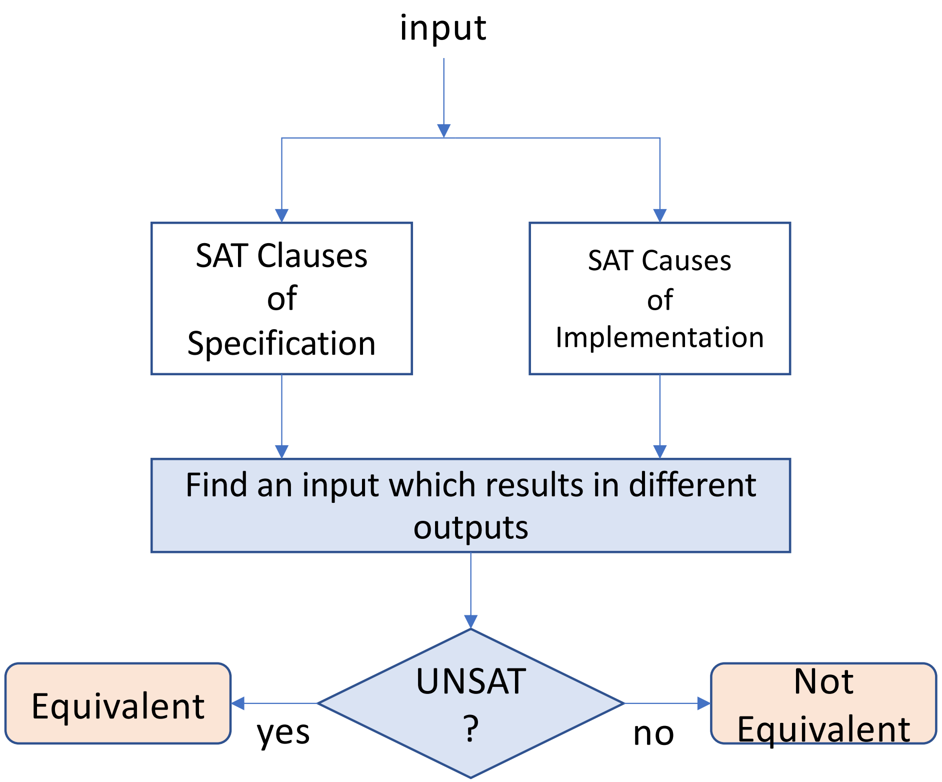
$$((\neg in_i[127:64] \& k_i[127:64]) \mid (\neg k_i[127:64] \& in_i[127:64])) \mid\mid 0^{64},$$

$$\text{and } ((\neg in_i[127:64] \& k_i[127:64]) | (\neg k_i[127:64] \& in_i[127:64]))$$

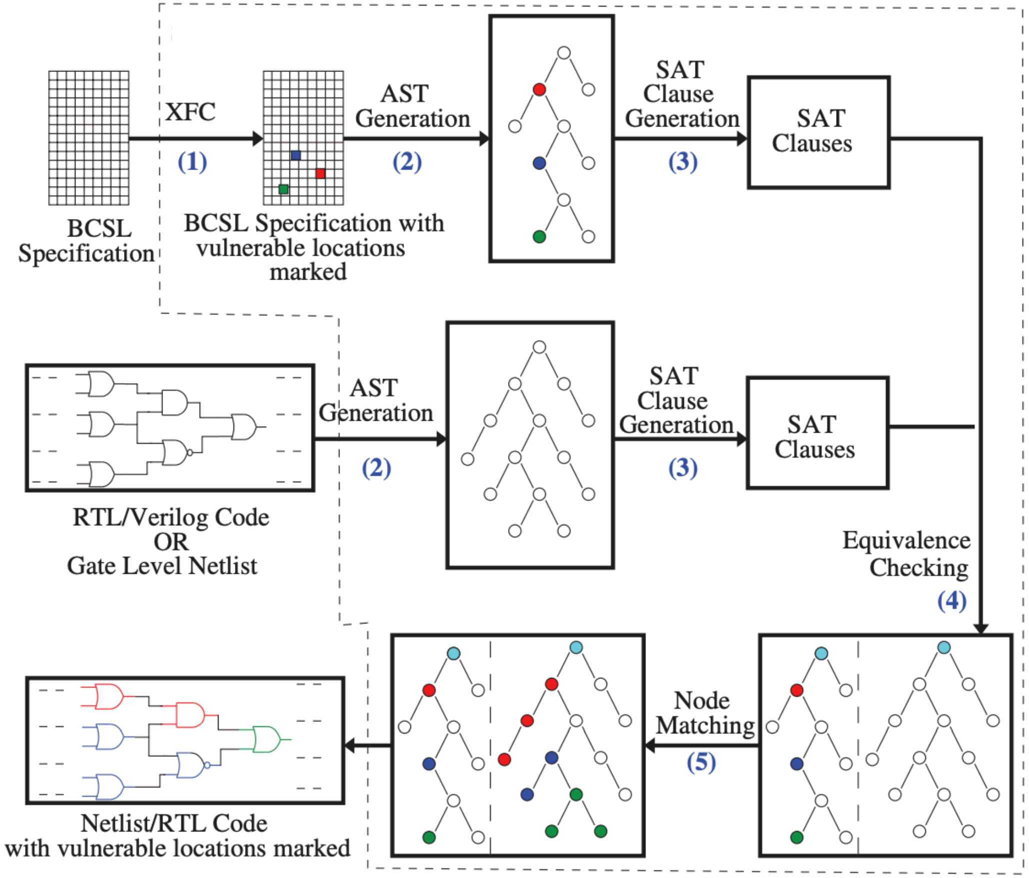
$$\mid\mid ((\neg in_i[63:0] \& k_i[63:0]) | (\neg k_i[63:0] \& in_i[63:0])),$$

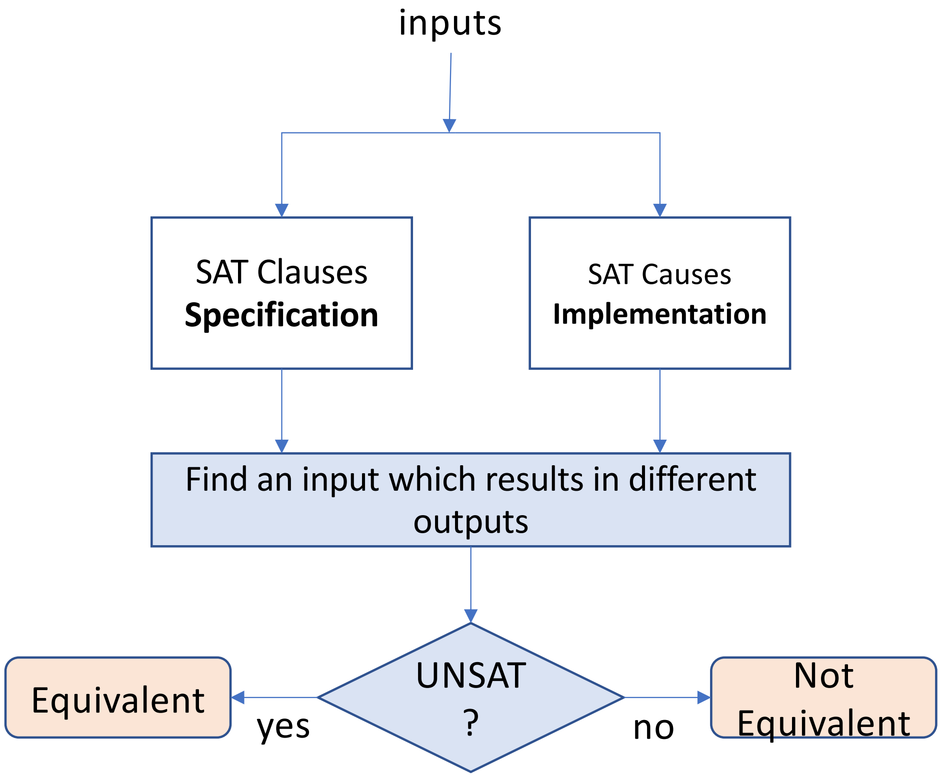
SOLOMON: An automated framework for detecting fault attack vulnerabilities in hardware





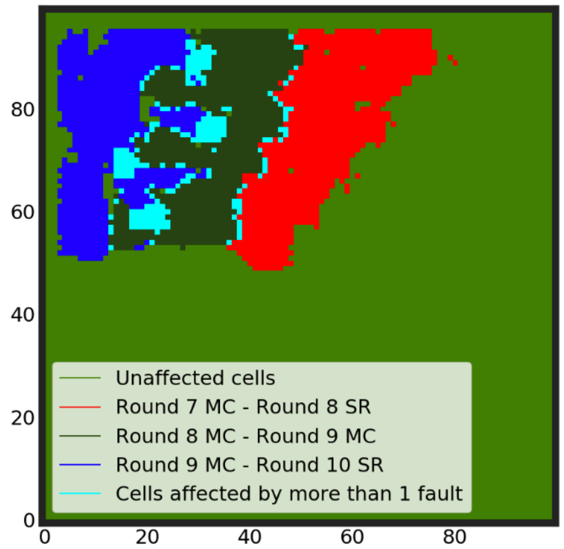
SOLOMON: An automated framework for detecting fault attack vulnerabilities in hardware



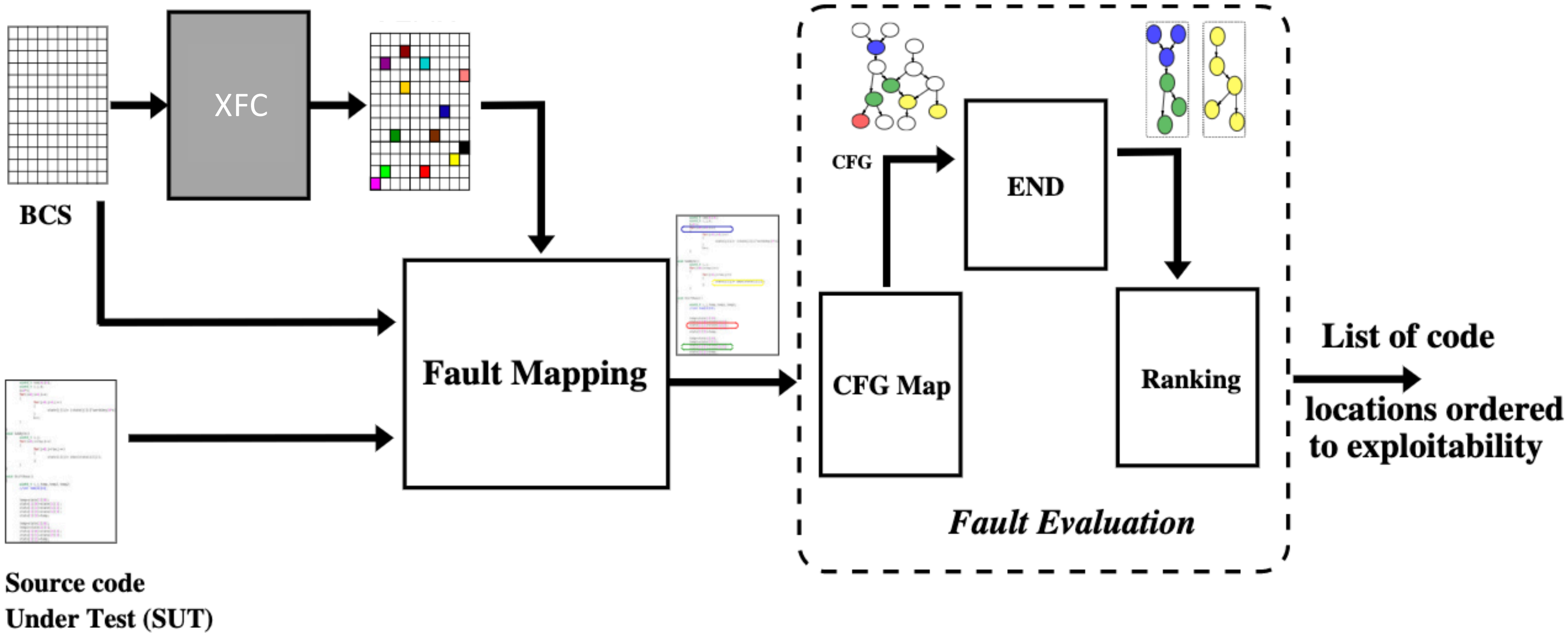


SOLOMON: An automated framework for detecting fault attack vulnerabilities in hardware

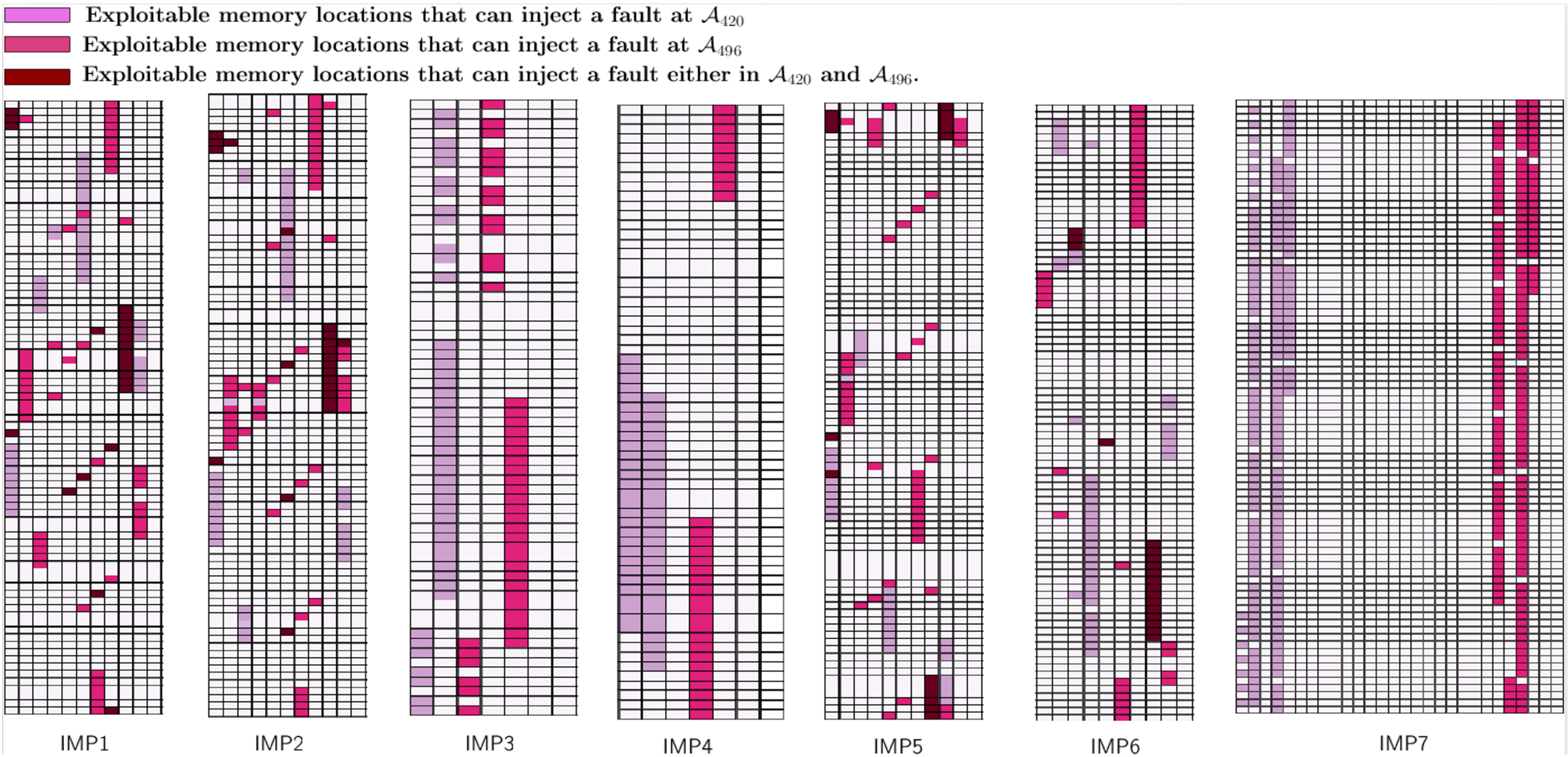
		$\mathbf{AES_C}$	$\mathbf{AES_L}$	**CLEFIA**	**Simon**
No. of AST nodes	*spec*	50379		28286	3961
	imple	2046	46041	3832	141
Execution time of each step (in sec)	**(1)**	0.02	0.02	0.02	0.06
	(2) *spec*	1.92		1.08	0.17
	(2) *imple*	0.12	1.75	0.17	0.03
	(3) *spec*	5.72		3.31	0.44
	(3) *imple*	0.23	5.54	0.56	0.36
	(4)	437.06	2.64	46.40	0.02
	(5)	13.8	0.92	136.7	0.24
	Total	458.87	18.51	188.24	1.32
Fault Vulner--ability analysis	**Fault location**	7 MixColumns to 8 ShiftRows		13 DXor to 14 SubByte	30 Rot_2 to 30 Concat
	Verilog lines	365	4173	609	24
	Gates	4590 (11.56%)	10477 (9.85%)	5324 (5.53%)	52 (2.83%)



FEDS: An Automated framework for detect fault attack vulnerabilities in **Software**

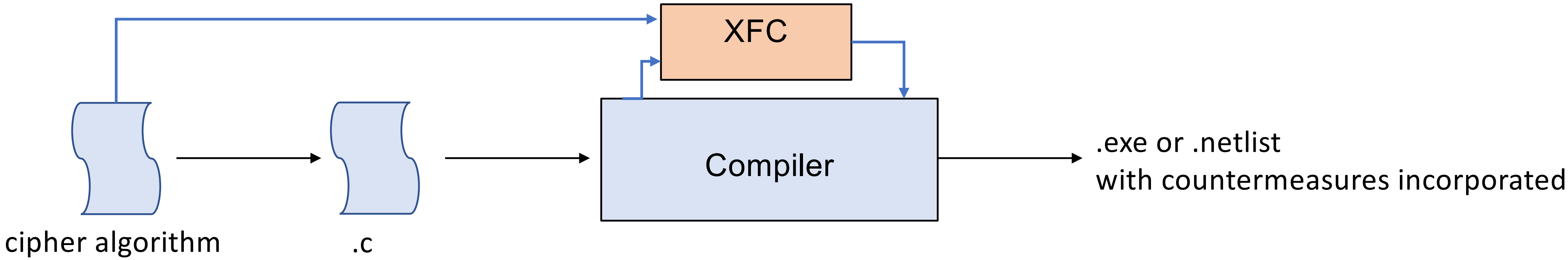


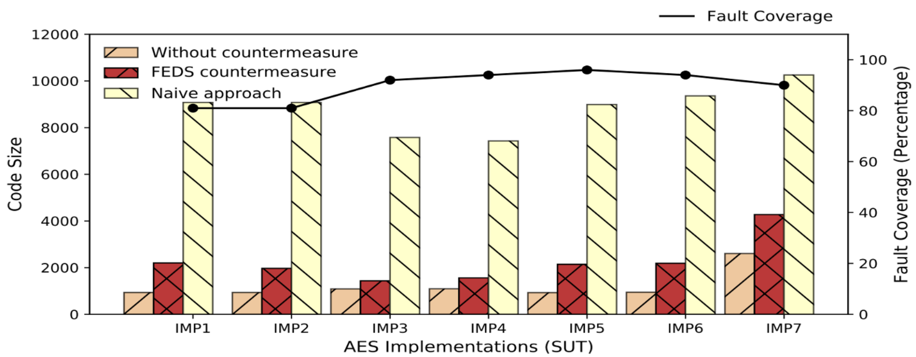
Fault Evaluation of 7 different implementations of AES



Automated Countermeasures using XFC

Fault Attack aware Compilers





Thank you for your attention

Source code: https://bitbucket.org/casl/faultanalysis/src/master/