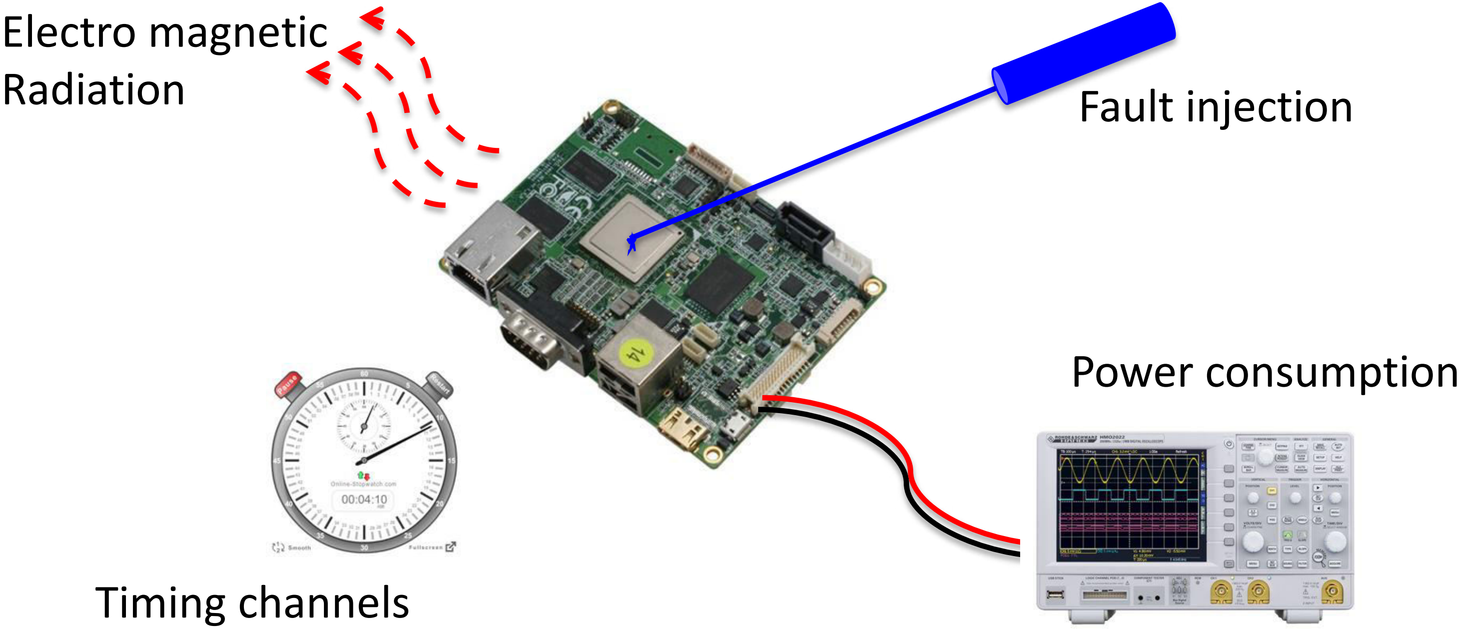
Automatic Synthesis of Fault Attack Resistant Cipher Implementations

Chester Rebeiro

IIT Madras

Side Channel Analysis



Preventing Side Channel Attacks is Difficult

overheads

Platform / Compiler Specific

Programming Language Specific

Naïve implem
have signific
performanc

may introd
Vulnerabilities

Cannot be implemented by your average Joe

Automatic Synthesis of protected implementations

Block Cipher Specification Language

```
h begin i
    h lookups i
        SBOX : { 0x63 , 0x7c , 0x77, 0x7b, · · · }
        KEY : { 0x54, 0x68, 0x61, 0x74, · · · }
    h /lookups i
    h operations i
            h func i  h MUL2 ( a ) i
                h h : { a : RS ( a, 7 ) } i
                h t : { a : LS ( a, 1 ) } i
                h n : { h : MUL ( h, 00x1b0 ) } i
                h m : { ( n, t ) : XOR ( n, t ) } i
                ret m
            h / func i
            · · · · · · · · ·
    h /operations i
    · · · · · · · · ·
    h F 2 i  h nonlinear i  h SUBBYTE i  h
        h F 2[1] : { F 1[1] : LKUP ( F 1[1], SBOX ) } i
        · · · · · · · · ·
        h F 2[16] : { F 1[16] : LKUP ( F 1[16], SBOX ) } i
    /i
    h F 3 i  h linear i  h PERMUTE i  h
        h F 3[1] : { F 2[1] } i
        · · · · · · · · ·
        h F 3[16] : { F 2[12] } i
    /i
    h F 4 i  h linear i  h MDS i  h
        h F 4[1] : { ( F 3[1], F 3[2], F 3[3], F 3[4] ) :
        XOR ( XOR ( MUL2 ( F 3[1] ), MUL3 ( F3[2] ) ), XOR ( F 3[3], F 3[4] ) ) } i
        · · · · · · · · ·
        h F 4[16] : { ( F 3[13], F 3[14], F 3[15], F 3[16] ) :
        XOR ( XOR ( MUL2 ( F 3[16] ), MUL3 ( F 3[23] ) ), XOR ( F 3[14], F 3[15] ) ) } i
    /i
    · · · · · · · · ·
h end i
```

HTML like language that captures basic functionality of the cipher

- operations
- information flow

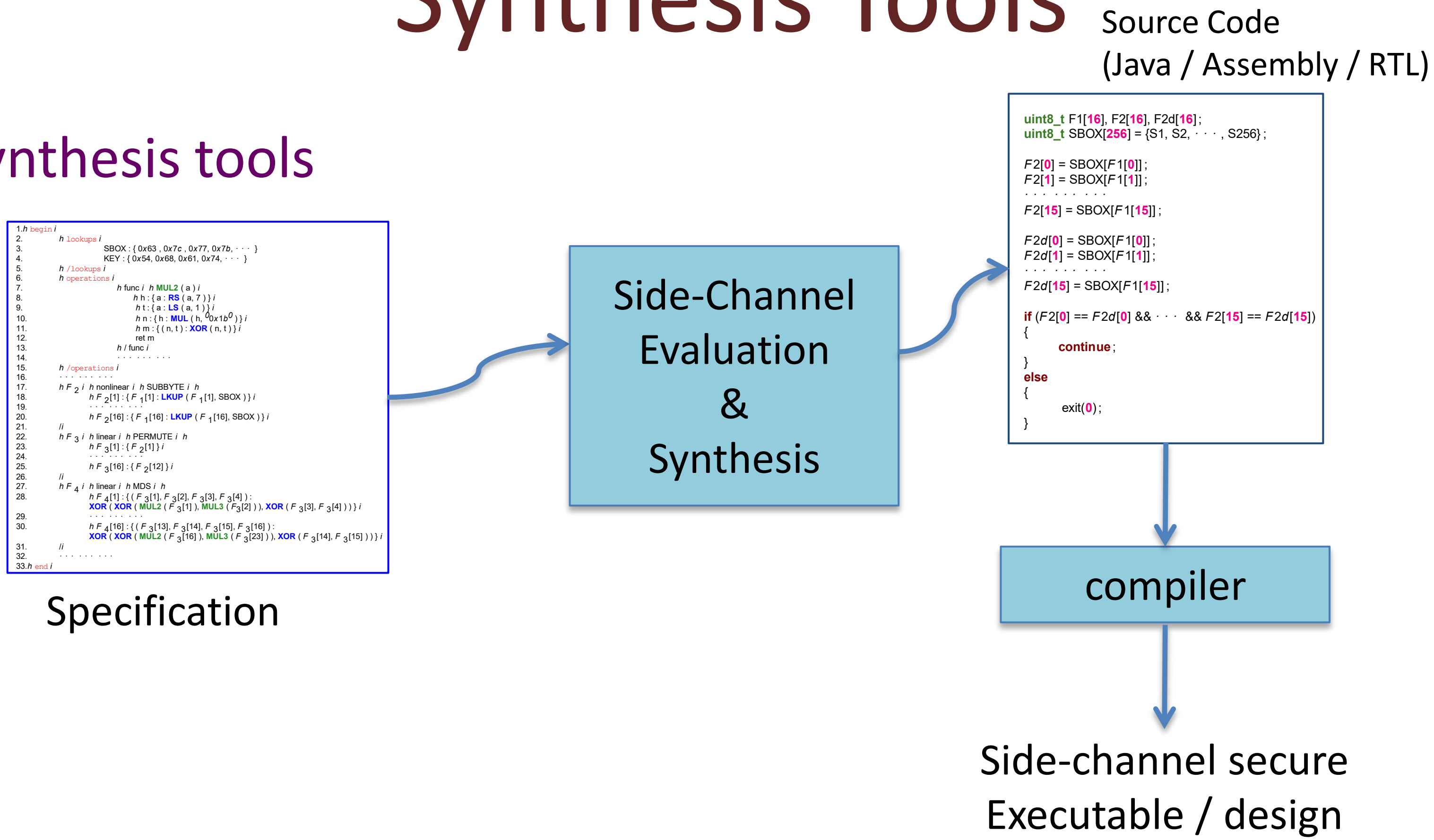
Just need one per cipher

Platform independent

Programming language independent

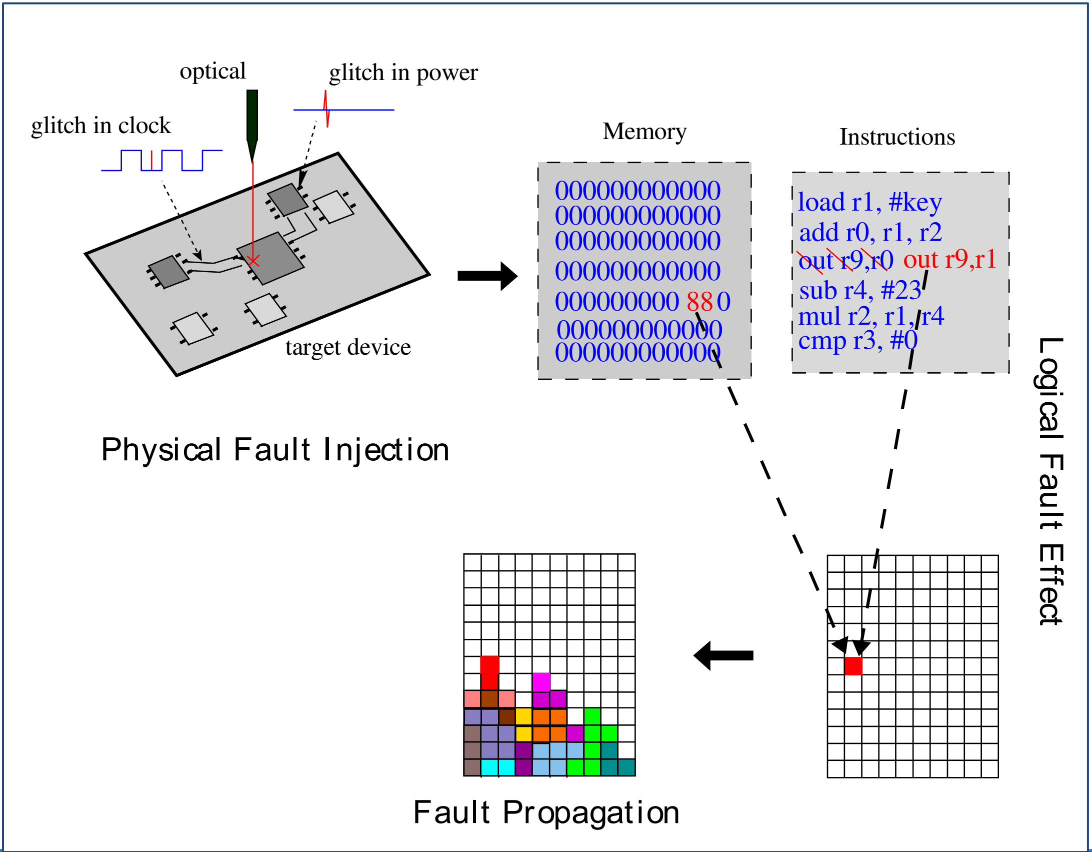
Synthesis Tools

Synthesis tools



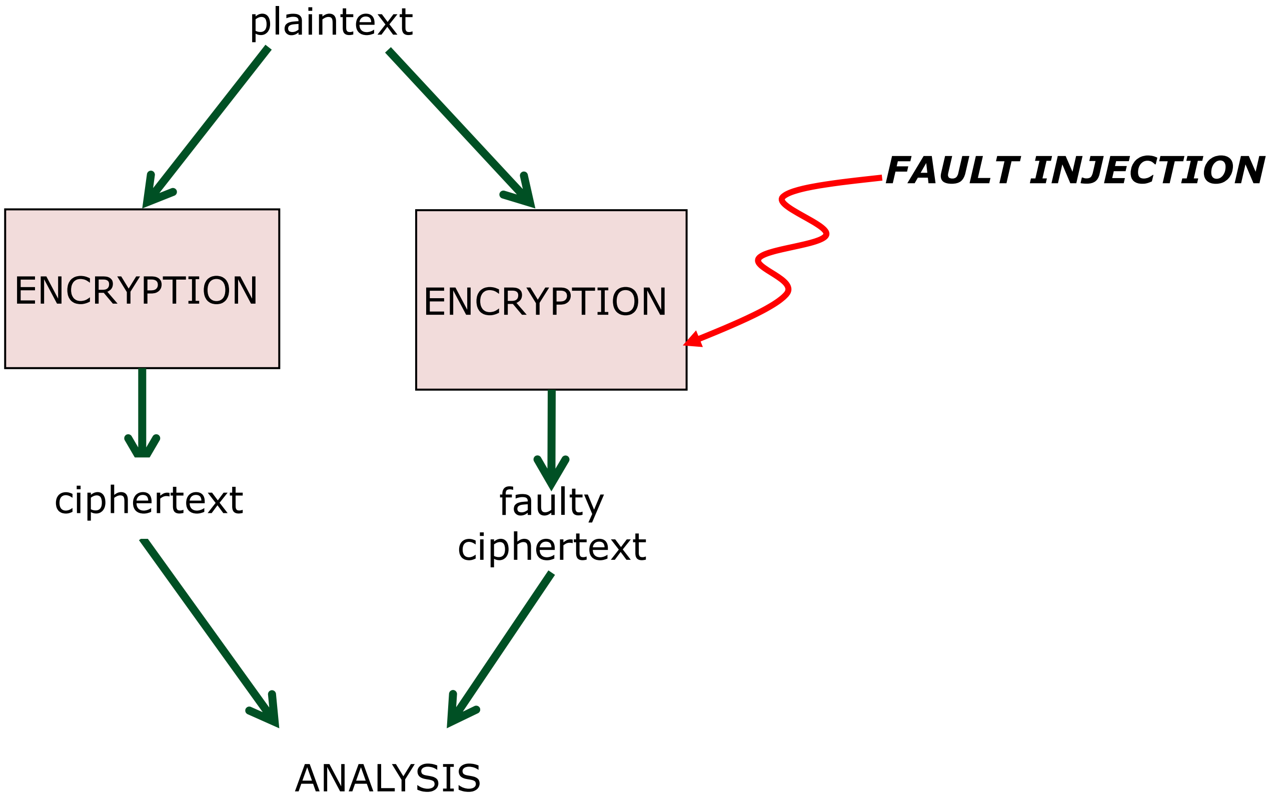
Automatic Synthesis of Fault Attack Resistant Block Cipher Implementations

Fault Injection in Block Ciphers

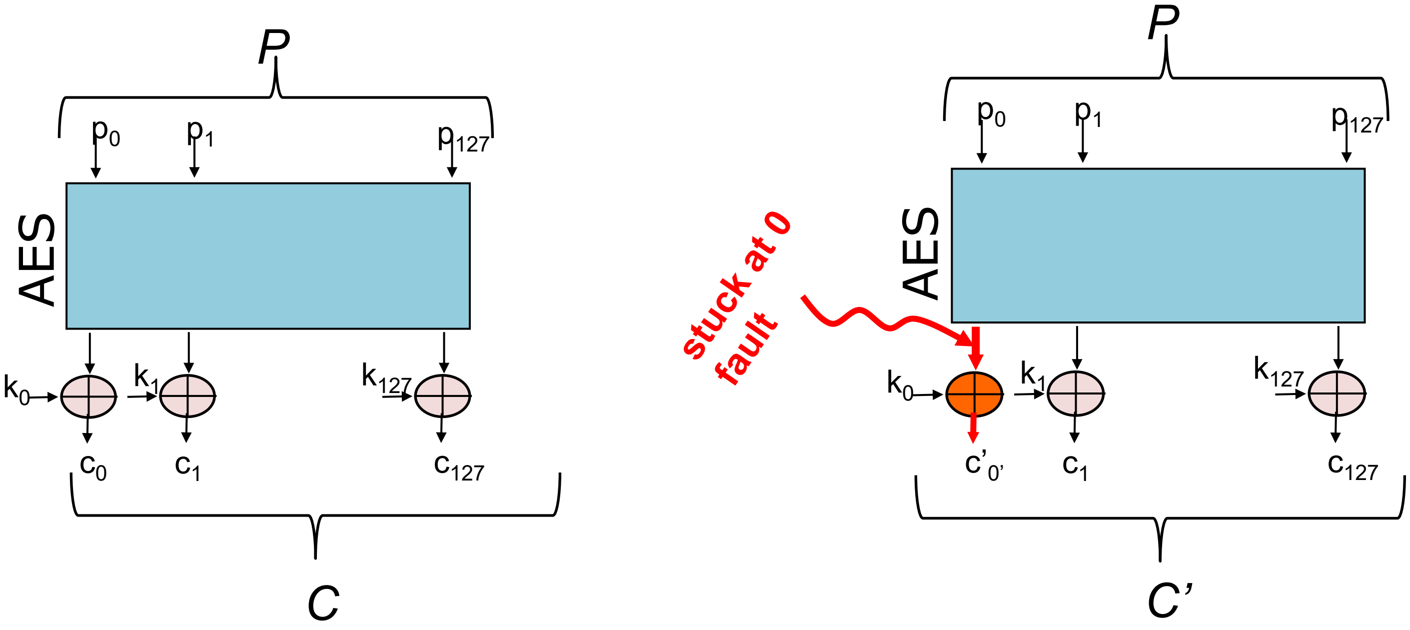


Random single byte fault injection

Differential Fault Attacks



A Simple AES Fault Attack



$$k_0 = c_0^{'}$$

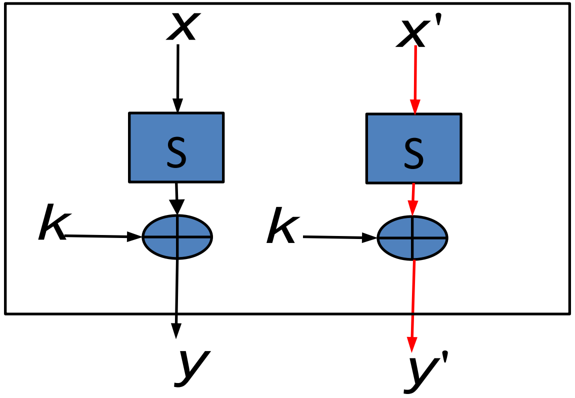
Many more stronger fault attacks are possible

Central Idea in DFA

Fault attackers look to solve equations in one of the following form

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

$$x \oplus x' =$$



where x and x' are known and k and are unknown.

Iterate over all possible values of and identify those that satisfy the above equations.

The number of solutions depends on the properties of the S-box.

Central Idea in DFA

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

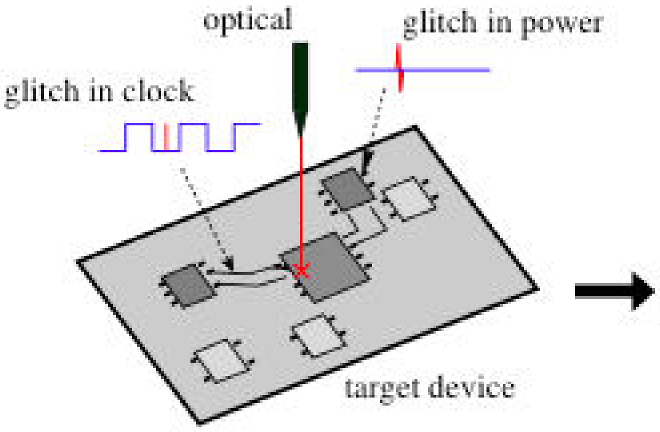
$$
\begin{aligned}
&S^{-1}(y_1 \oplus k_1) \oplus S^{-1}(y_1' \oplus k_1) = g_1(\delta) \\
&S^{-1}(y_2 \oplus k_2) \oplus S^{-1}(y_2' \oplus k_2) = g_2(\delta) \\
&\vdots \qquad\quad \vdots \qquad \vdots \qquad\quad \vdots \qquad\qquad \vdots \\
&S^{-1}(y_n \oplus k_n) \oplus S^{-1}(y_n' \oplus k_n) = g_n(\delta)
\end{aligned}
$$

where $g_1(\delta),\ g_2(\delta),\cdots,g_n(\delta)$ are linear functions in

Only key tuples $(k_1, k_2, \cdots, k_n)$ that satisfy all n equations are potential candidates

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

summarize



Involving sbox

Linear functions

$$
\begin{array}{l}
S^{-1}(y_1 \oplus k_1) \oplus S^{-1}(y_1^{'} \oplus k_1) = g_1(\delta) \\
S^{-1}(y_2 \oplus k_2) \oplus S^{-1}(y_2^{'} \oplus k_2) = g_2(\delta) \\
\vdots \qquad \vdots \qquad \vdots \qquad \vdots \qquad \vdots \\
S^{-1}(y_n \oplus k_n) \oplus S^{-1}(y_n^{'} \oplus k_n) = g_n(\delta)
\end{array}
$$

More the better

Inject Fault

Solve Equations

Online complexity : #faults are needed to retrieve the key
Offline complexity : Search space for finding the keys

Differential Fault Attacks on AES

Fault Injected	#faults (online complexity)	Offline complexity
in first round	requires 128 faults	NIL
in last round	requires 128 faults	NIL
in 9th round	requires 4 faults (each fault derives 32 bits of key)	256
in 8th round	Requires 1 fault	256
all other rounds	Not exploitable	N/A

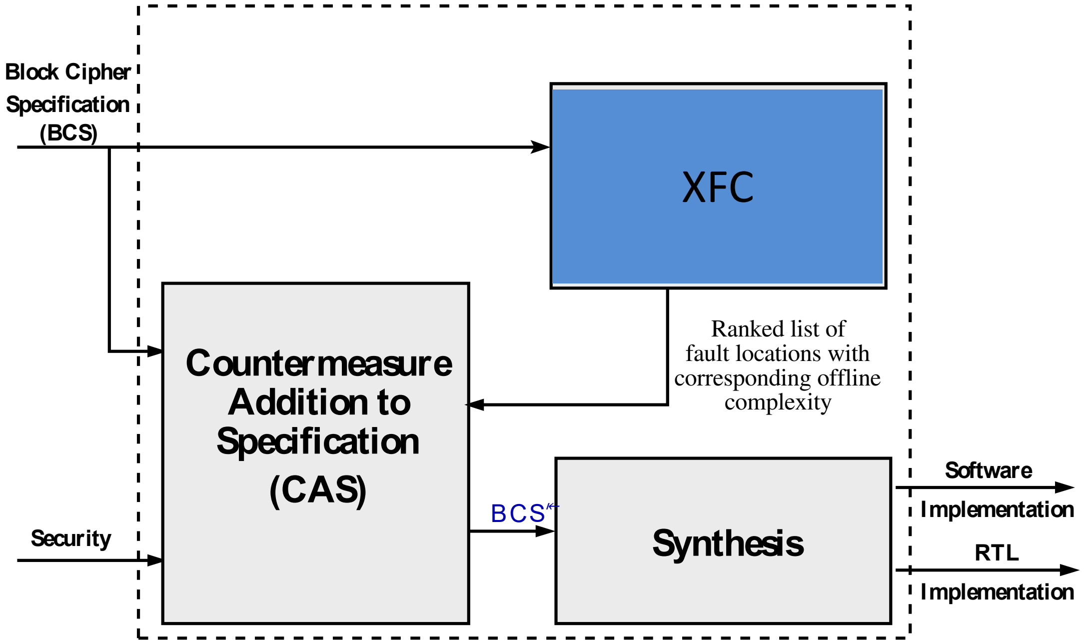
A majority of the faults are unexploitable

Naïve countermeasures do not consider the online or offline attack complexity requirements, thus significant overheads

SAFARI

Automatic Synthesis of Fault Attack Resistant Block Cipher Implementations

XFC: Exploitable Fault Characterization

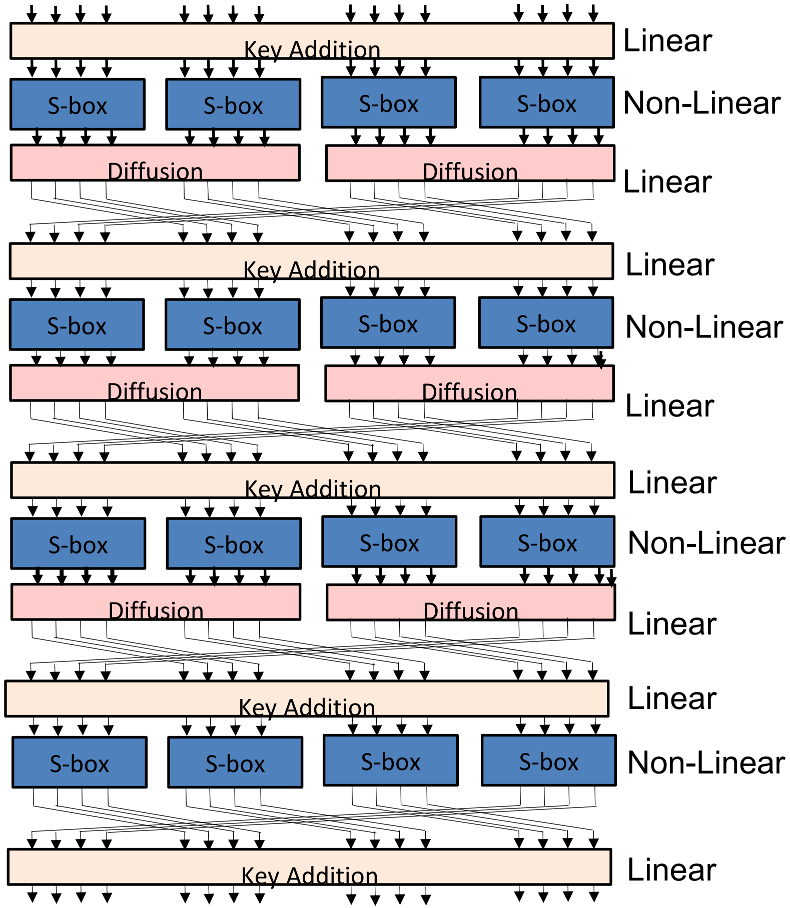


XFC


Block Cipher Specification
fault position
Fault Model
XFC
Derivable Round Keys
Complexity of Attack

Fault Propagation Phase

A Typical Block Cipher



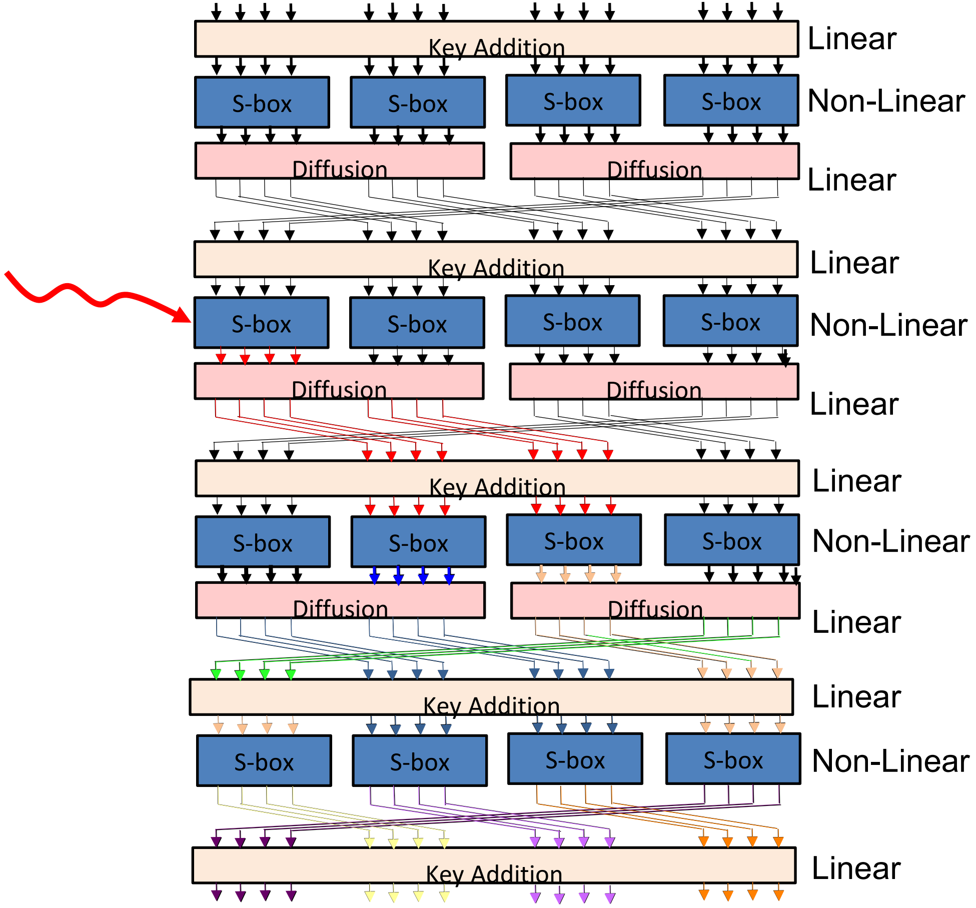
Fault Propagation Phase

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

1. If two bytes of different colors are combined, change the color.



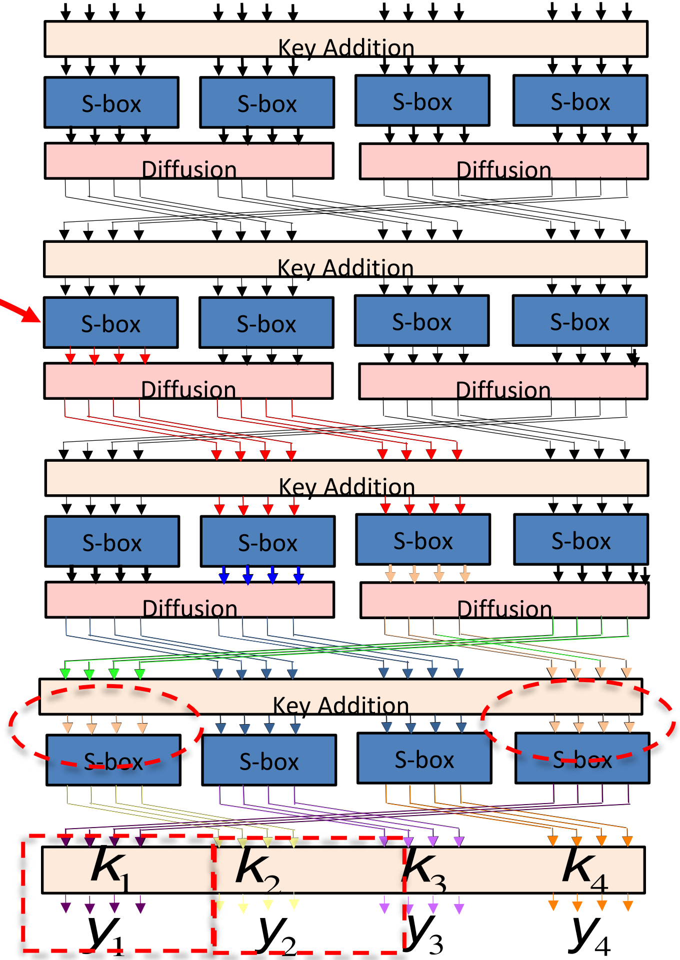
Same colors are linearly correlated
Different colors are not correlated

Key Determination Phase

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

For every possible value of
determine k_1 , k_2 that satisfy the equations.

The complexity is 2^4 ; the possible
values can take.

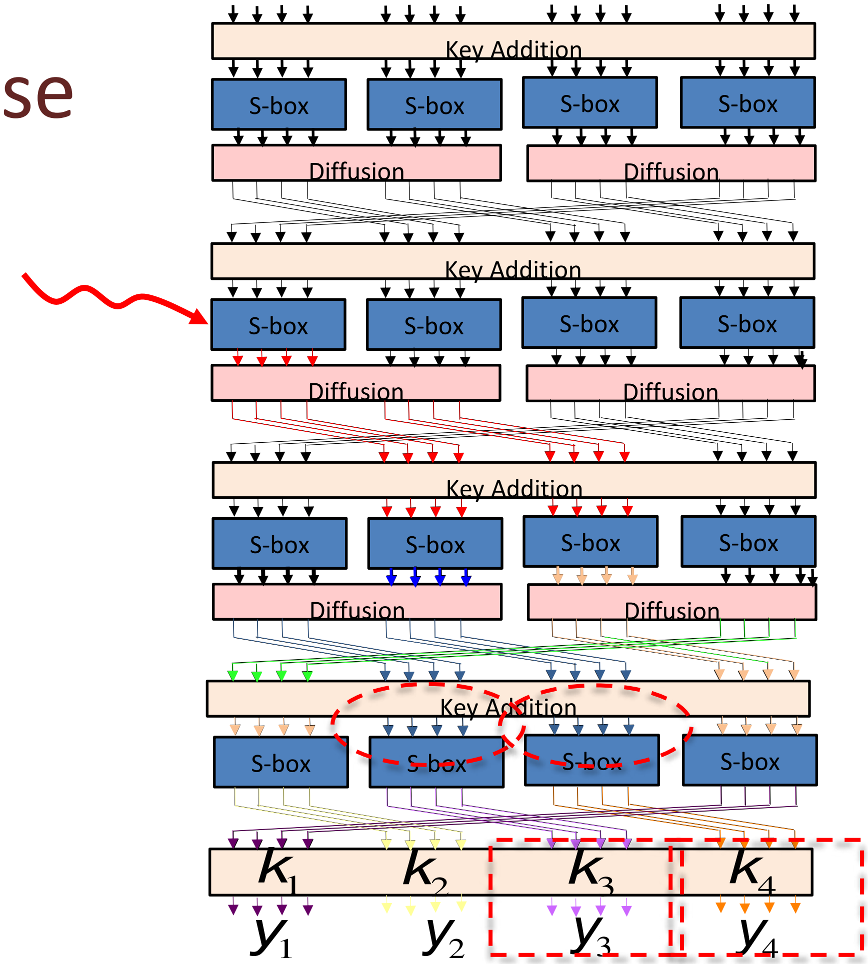


Key Determination Phase

$$S^{1}(y_3 \oplus k_3) \oplus S^{1}(y_3' \oplus k_3) = g_3(\quad)$$

$$S^{1}(y_4 \oplus k_4) \oplus S^{1}(y_4' \oplus k_4) = g_4(\quad)$$

Can be used to determine (k_3, k_4)

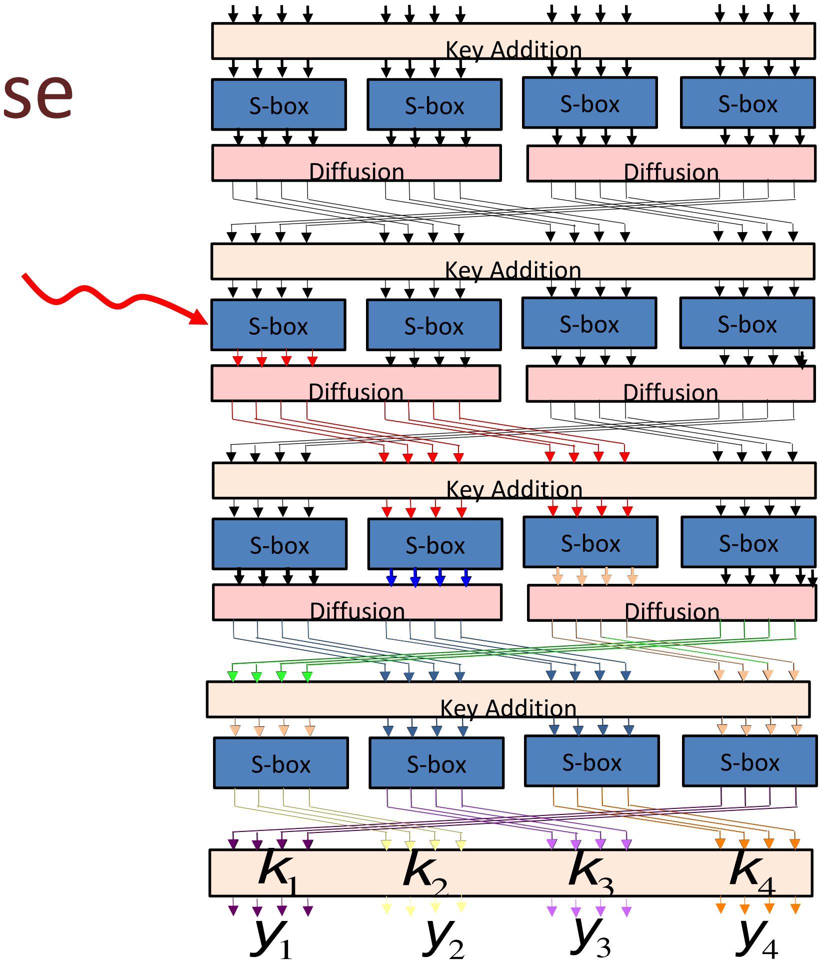


Key Determination Phase

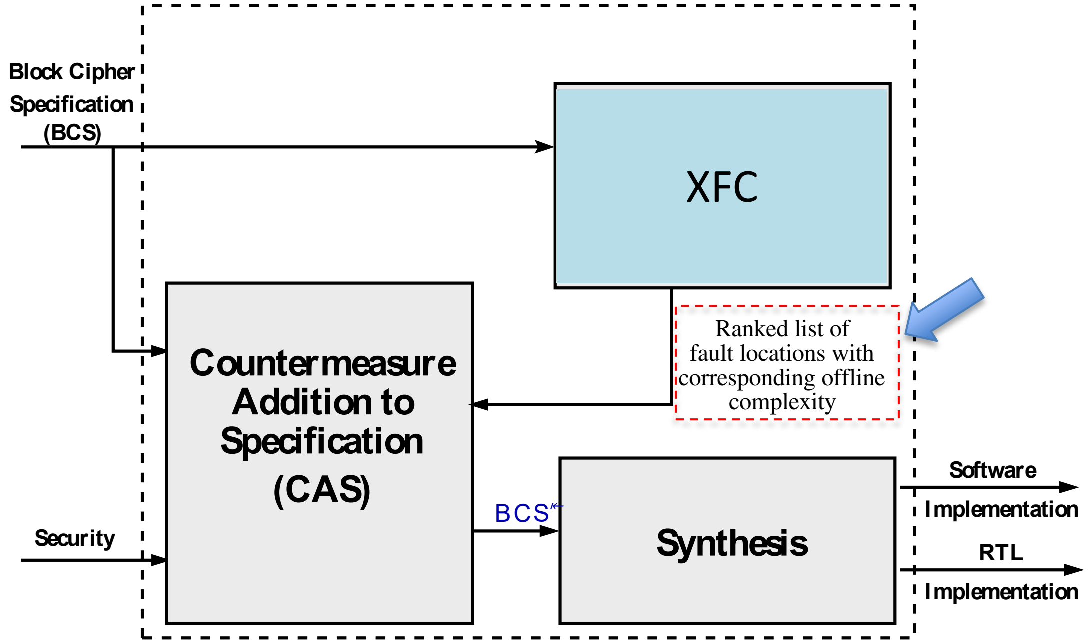
We can thus determine keys (k_1, k_2, k_3, k_4) and, based on the S-Box properties, we can computer the offline complexity

Continue this process upwards toward the location of the fault eventually getting a result As follows:

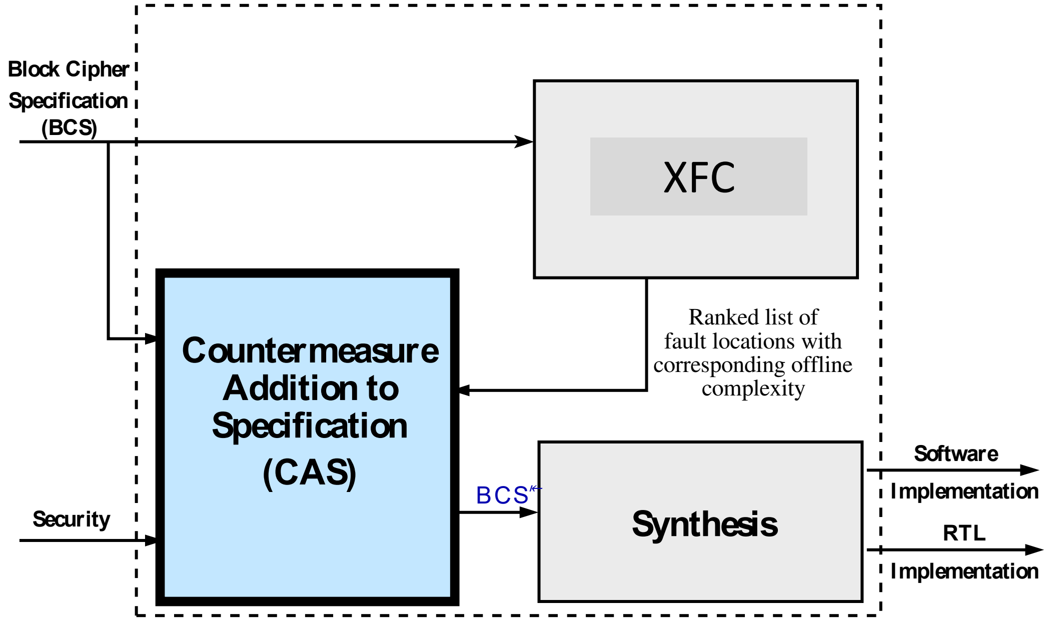
Fault at location x gives keys (k_1, k_2, k_3, k_4) with an offline complexity of



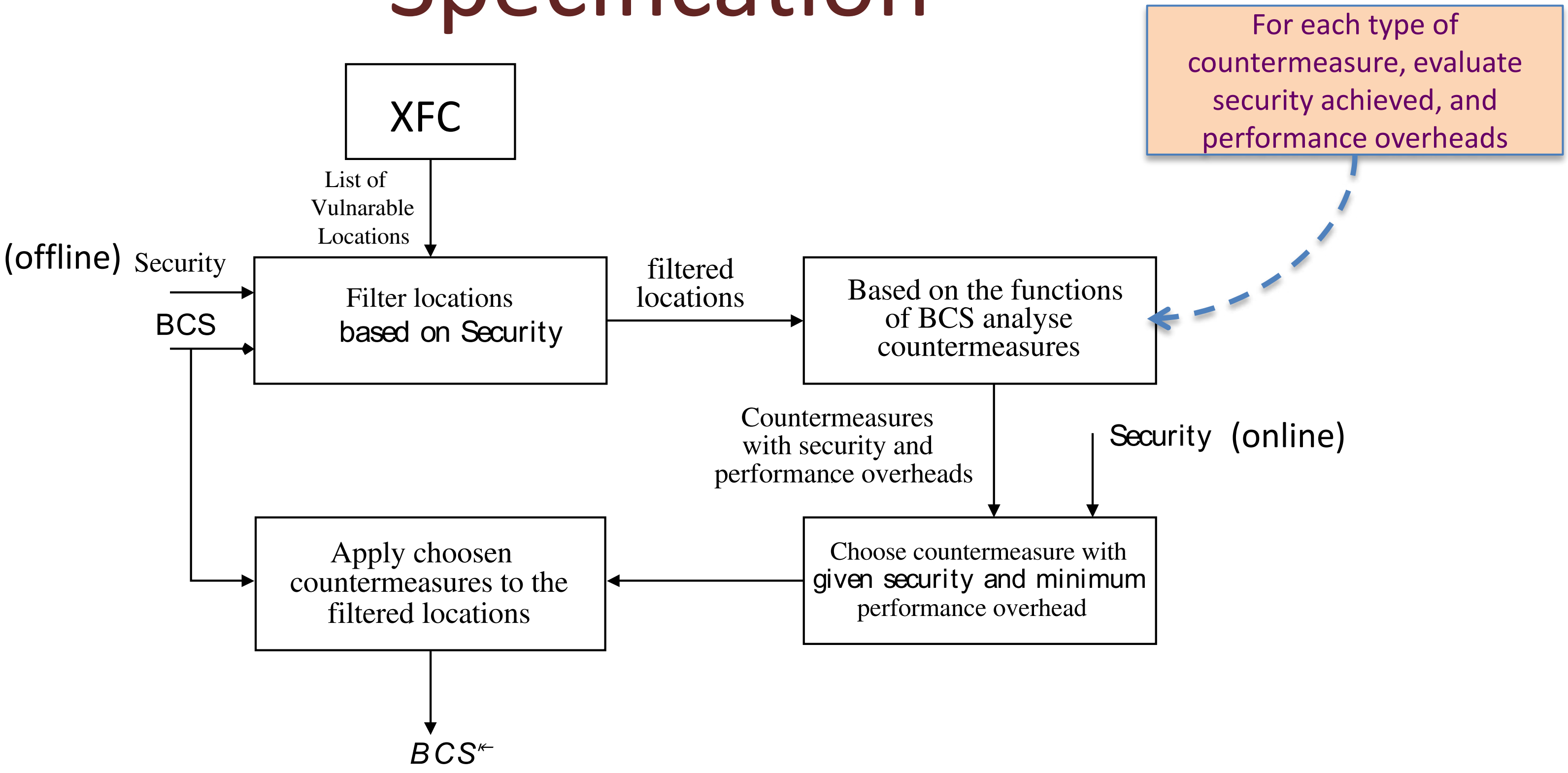
XFC: Exploitable Fault Characterization



Countermeasure Addition to Specification



Countermeasure Addition to Specification



Analysis of Redundancy

Security Analysis

If the probability of injecting a fault f is p_f, then the probability with which an attack is successful (p_s) is defined as :

$$p_s = \sum_{i=1}^{255} p_f \times p_f = \sum_{i=1}^{255} {p_f}^2 \ .$$

Performance Overhead

- Overhead is estimated in terms of time and area from $BCS^{\leftarrow}$.
- For Software synthesis, number of extra clock cycles required is calculated.
- If the target device is known then we know the number of clock cycles taken by each of the primitive operations. Hence total time can be estimated
- For Hardware synthesis overhead estimated through number of extra gates used.

Countermeasure Evaluation

For Software

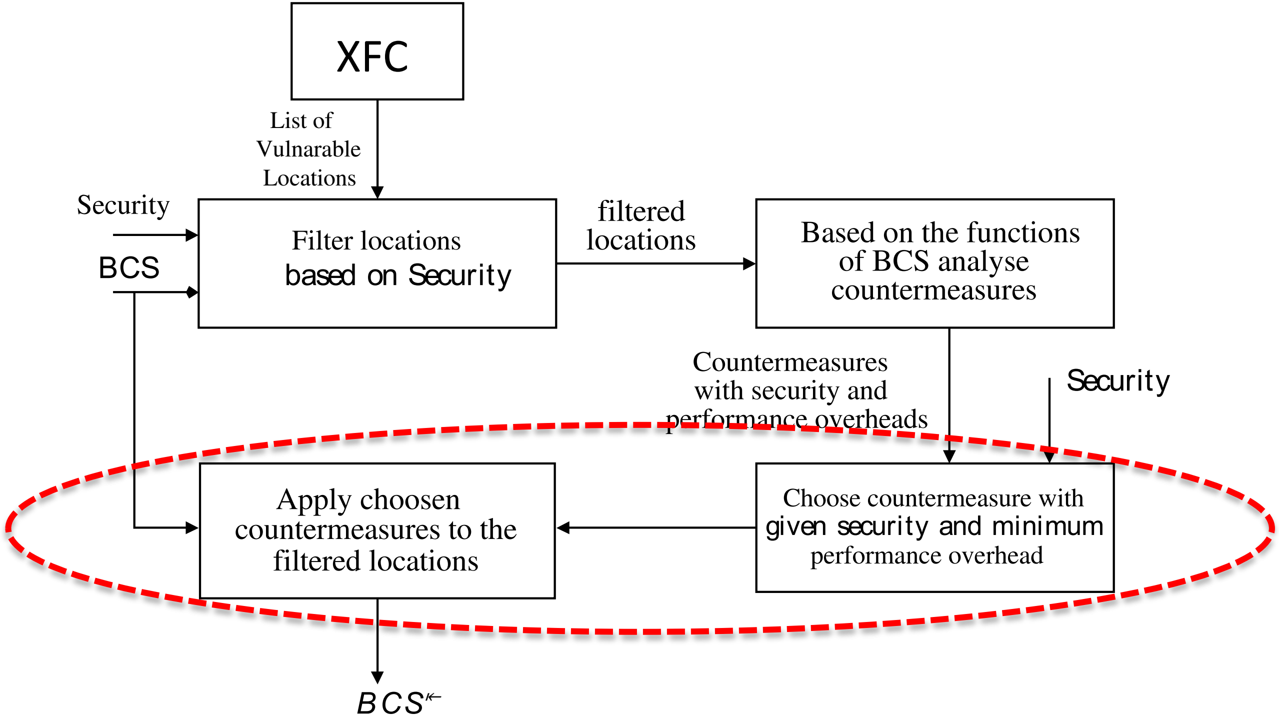
Countermeasures	Security	Function	Memory Overhead (in bytes)	T_e	*Memory* ⇥ T_e
Redundancy	1/ 255	ARK	64	96	6144
	1/ 255	SB	256	64	16384
	1/ 255	MC	-	400	400
1 Bit Parity	1/ 2	ARK	-	256	256
	1/ 2	SB	32	256	8192
	63/ 255	MC	-	448	448
2 Bit Parity	1/ 4	ARK	-	288	228
	1/ 4	SB	64	272	17409
	15/ 255	MC	-	960	960
4 Bit Parity	1/ 16	ARK	-	192	192
	1/ 16	SB	128	224	28672
	1/ 255	MC	-	976	976

Countermeasure Evaluation

For Hardware

Countermeasure	Security	Function	XOR	AND	OR	Others
Redundancy	1/ 255	ARK	256	0	128	16, 8 Bit Register
	1/ 255	SB	128	0	128	8 Bit 256 → 1 MUX
	1/ 255	MC	768	0	128	-
1 Bit Parity	1/ 2	ARC	128	16	15	-
	1/ 2	SB	128	16	15	8 Bit 16 → 1 MUX
	63/ 255	MC	176	16	15	-
2 Bit Parity	1/ 4	ARC	128	32	31	-
	1/ 4	SB	128	32	31	8 Bit 32 → 1 MUX
	15/ 255	MC	432	32	31	-
4 Bit Parity	1/ 16	ARK	128	64	63	-
	1/ 26	SB	128	64	63	8 Bit 64 → 1 MUX
	1/ 255	MC	608	64	63	-

Countermeasure Addition to Specification



Modified Block Cipher Specification

h F_2 i h nonlinear i h SUBBYTE i h

h F_2[1] : { F_1[1] : **LKUP** (F_1[1], SBOX) } i

h F_2[2] : { F_1[2] : **LKUP** (F_1[2], SBOX) } i

. . . .

h F_2[16] : { F_1[16] : **LKUP** (F_1[16], SBOX) } i

/i

//Adding redundant computation of F_2

h $F_2^{\leftarrow}$ i h nonlinear i h SUBBYTE i h

h $F_2^{\leftarrow}$[1] : { F_1[1] : **LKUP** (F_1[1], SBOXd) } i

h $F_2^{\leftarrow}$[2] : { F_1[2] : **LKUP** (F_1[2], SBOXd) } i

. . . .

h $F_2^{\leftarrow}$[16] : { F_1[16] : **LKUP** (F_1[16], SBOXd) } i

/i

//Adding function to compare results of original and redundant function

h F_2^{d} i h linear i h CMP i h

h F_2^{d}[1] : { (F_2[1] , $F_2^{\leftarrow}$[1]) : **ECMP** (F_2[1] , $F_2^{\leftarrow}$[1]) } i

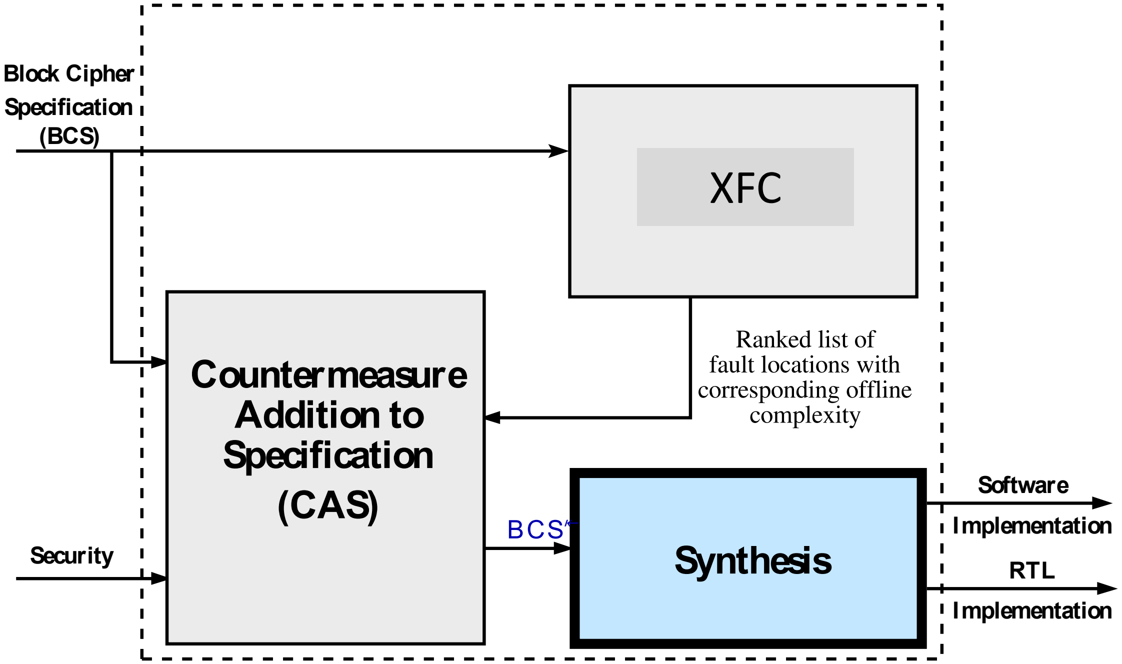
h F_2^{d}[2] : { (F_2[2] , $F_2^{\leftarrow}$[2]) : **ECMP** (F_2[2] , $F_2^{\leftarrow}$[2]) } i

. . . .

h F_2^{d}[16] : { (F_2[16] , $F_2^{\leftarrow}$[16]) : **ECMP** (F_2[16] , $F_2^{\leftarrow}$[16]) } i

/i

Synthesis



Block Cipher Specification (BCS)
XFC
Ranked list of fault locations with corresponding offline complexity
Countermeasure Addition to Specification (CAS)
Security
BCS'
Synthesis
Software Implementation
RTL Implementation

Synthesis

```
uint8_t F1[16], F2[16], F2d[16];
uint8_t SBOX[256] = {S1, S2, · · · , S256};

F2[0] = SBOX[F1[0]];
F2[1] = SBOX[F1[1]];
. . .  . . . .  . . .
F2[15] = SBOX[F1[15]];

F2d[0] = SBOX[F1[0]];
F2d[1] = SBOX[F1[1]];
. . .  . . . .  . . .
F2d[15] = SBOX[F1[15]];

if (F2[0] == F2d[0] && · · ·  && F2[15] == F2d[15])
{
        continue;
}
else
{
         exit(0);
}
```

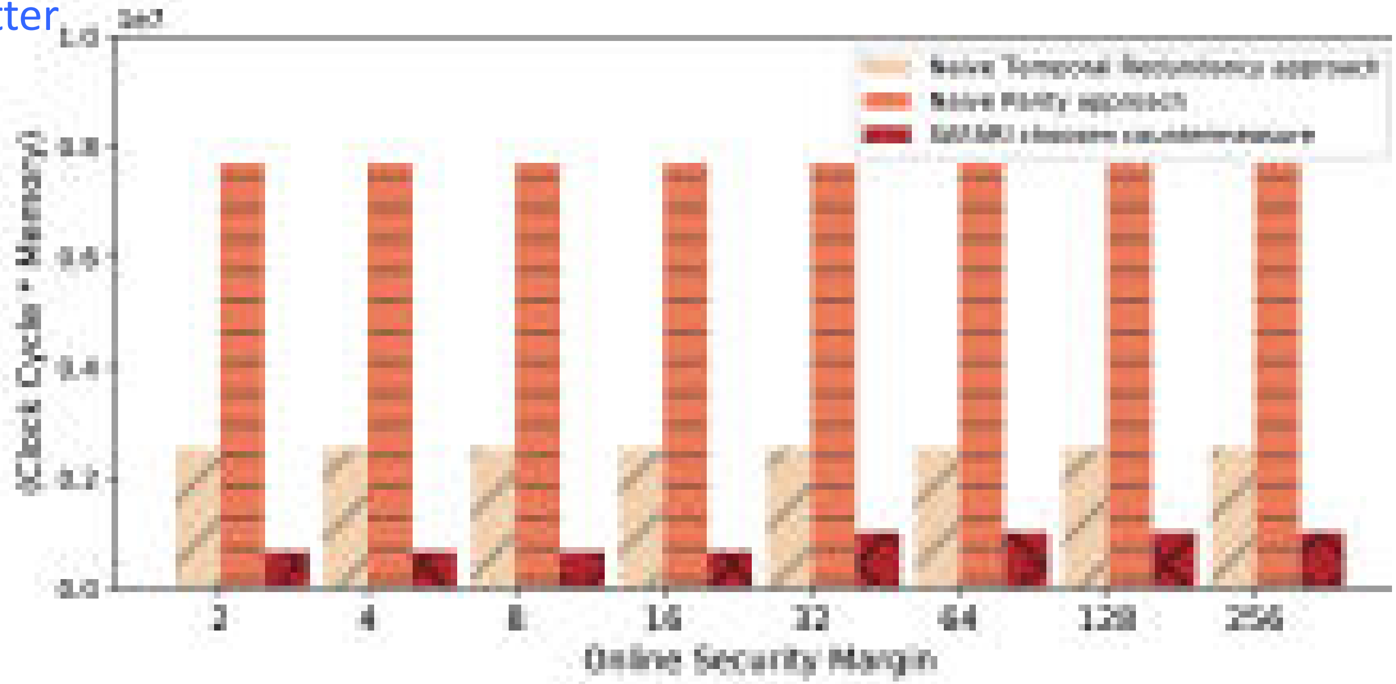
Synthesized C Code(Software)

```
wire[127 :0] F1, F2, F2d;
wire[15 :0] F2c;
SubByte SB1 (F1[127 : 120], F2[127 : 120]);
SubByte SB2 (F1[119 : 112], F2[119 : 112]);
. . .  . . . .  . . .
SubByte SB16 (F1[7 : 0], F2[7 : 0]);

SubByte DSB1 (F1[127 : 120], F2d[127 : 120]);
SubByte DSB2 (F1[119 : 112], F2d[119 : 112]);
. . .  . . . .  . . .
SubByte DSB16 (F1[7 : 0], F2d[7 : 0]);

F2c[15] = ECMP(F2[127 : 120], F2d[127 : 120]);
F2c[14] = ECMP(F2[119 : 112], F2d[119 : 112]);
. . .  . . . .  . . .
F2c[0] = ECMP(F2[7 : 0], F2d[7 : 0]);
```

Synthesized Verilog Code(Hardware)

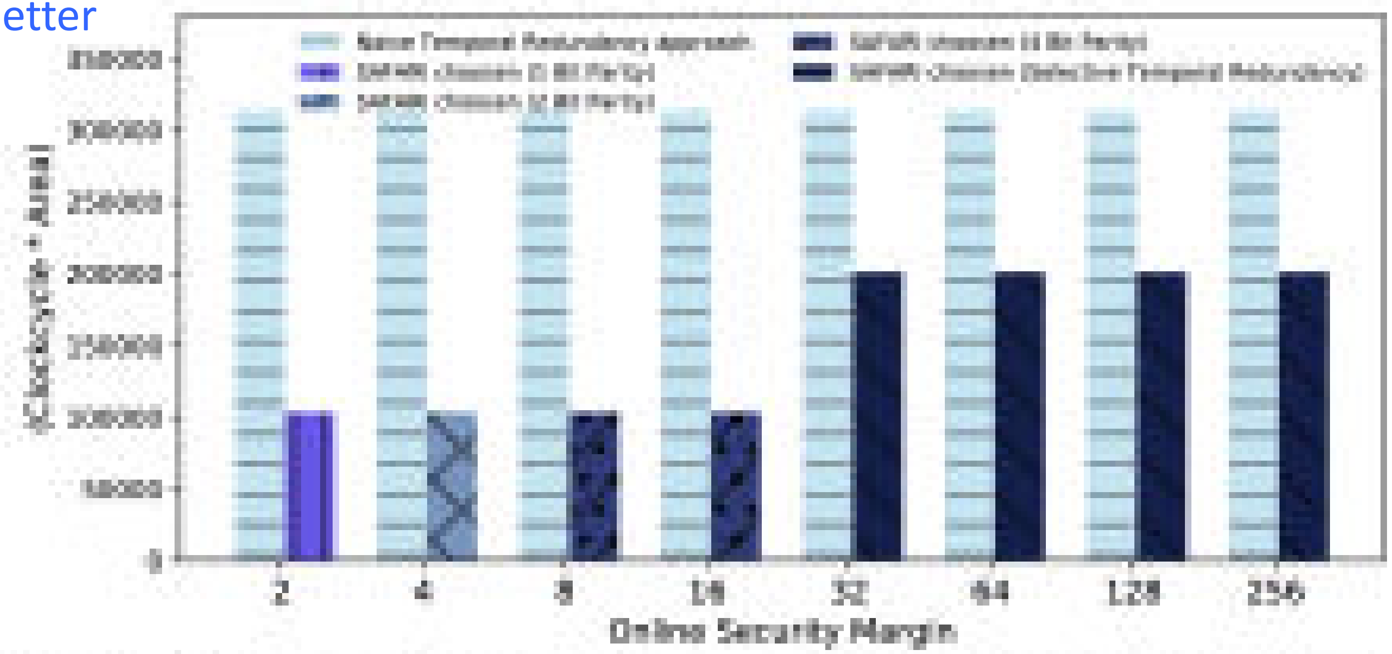
Results (Software)

Lower is better



Results (Hardware)

Lower is better



Conclusions

- First attempt to automatically synthesize fault attack resistant block cipher implementations
  - User configurable security
  - Reduced overheads
  - any block cipher, any device, any programming language
- Covers a wide range of ciphers and countermeasures
- Block Cipher Specification Language

- Limitations:
  - In software especially, the generated code is not very efficient for high performance processors.