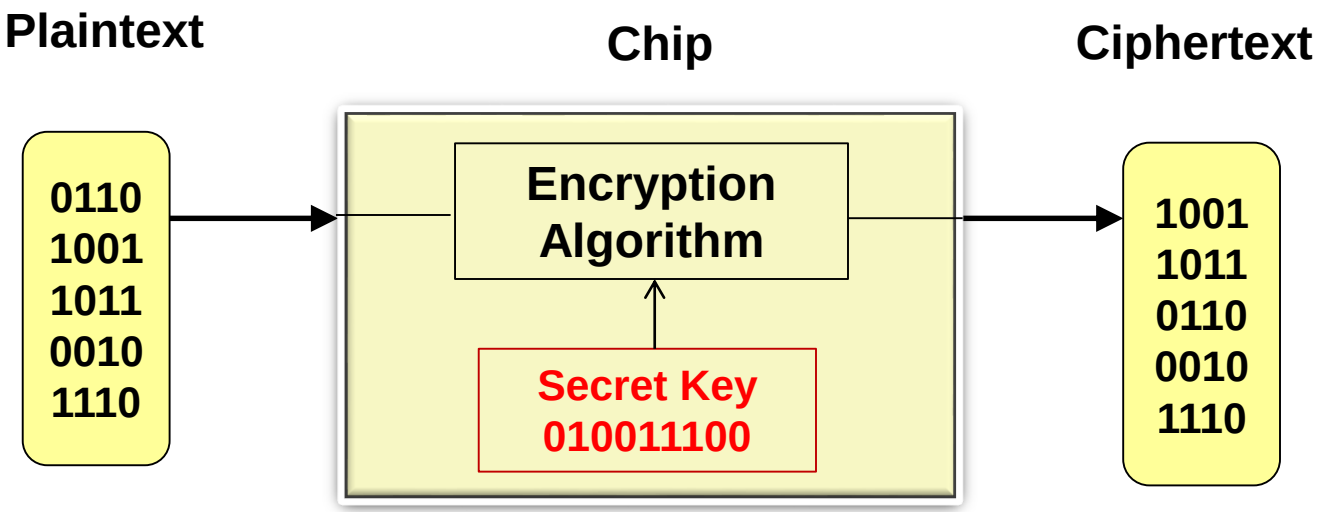
VirginiaTech
Invent the Future

Synthesis of Fault-Attack Countermeasures for Cryptographic Circuits

Hassan Eldib, Meng Wu, and **Chao Wang**

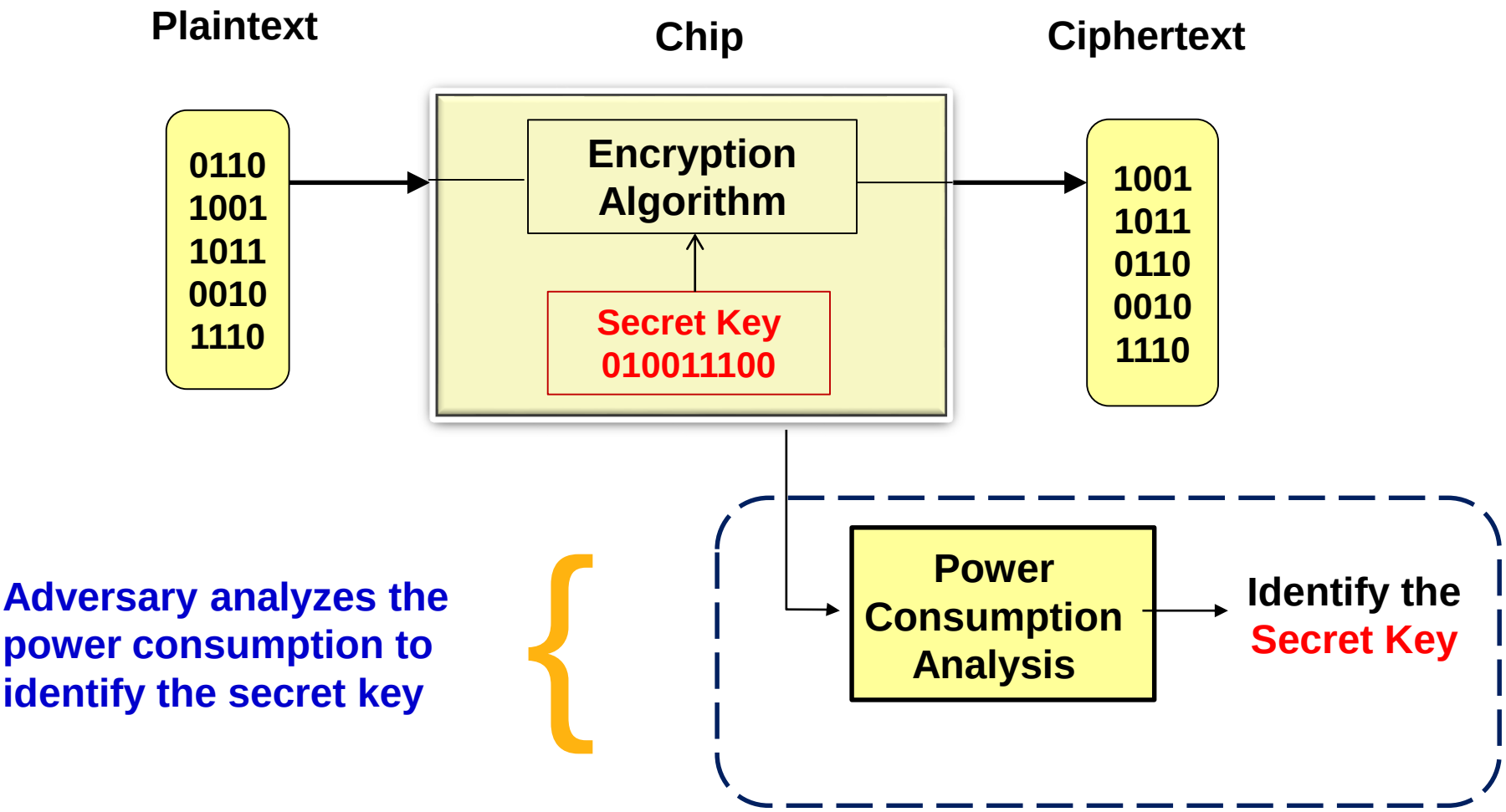
CAV, July 23, 2016

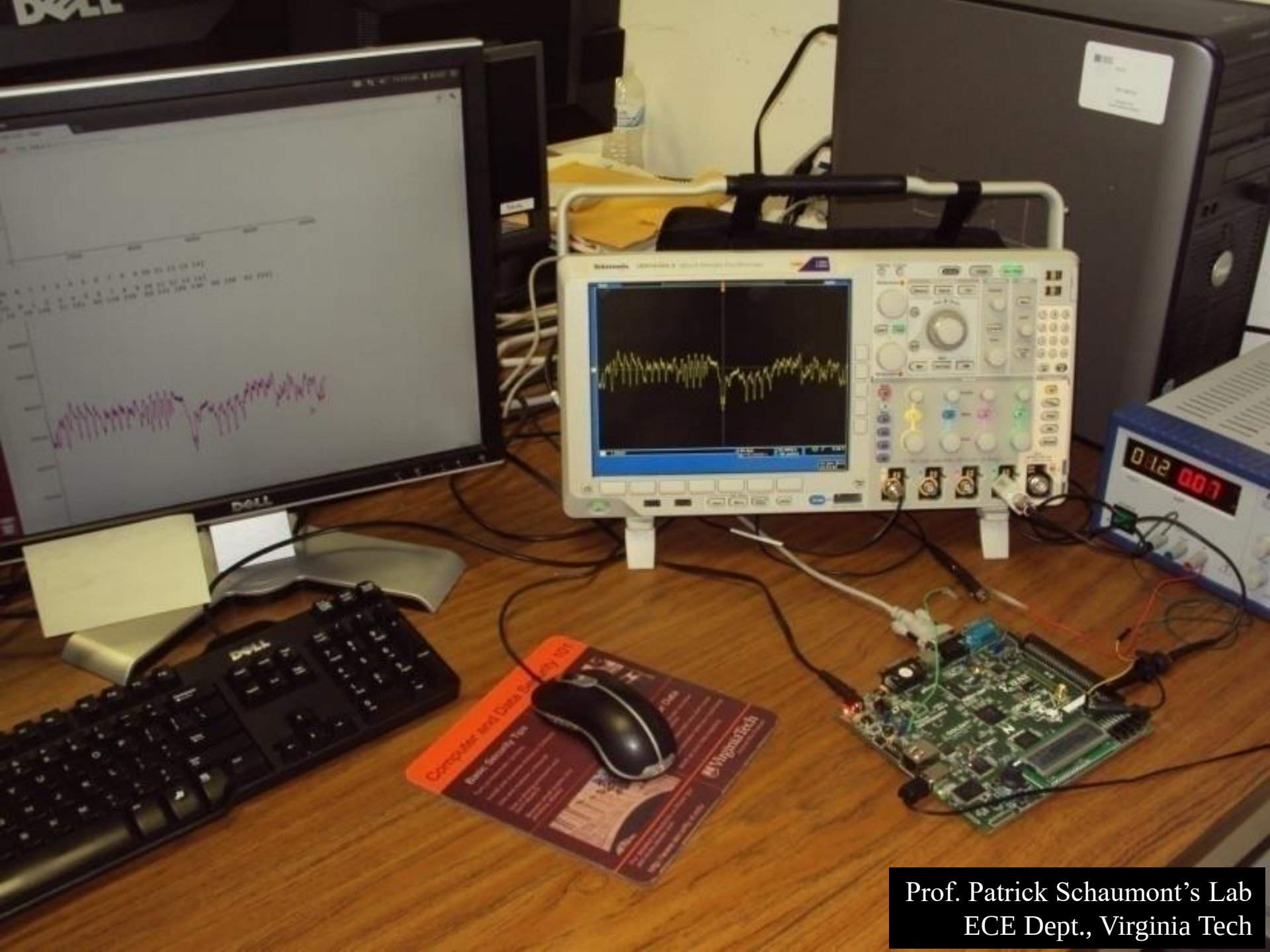
Cryptographic Algorithm: an example



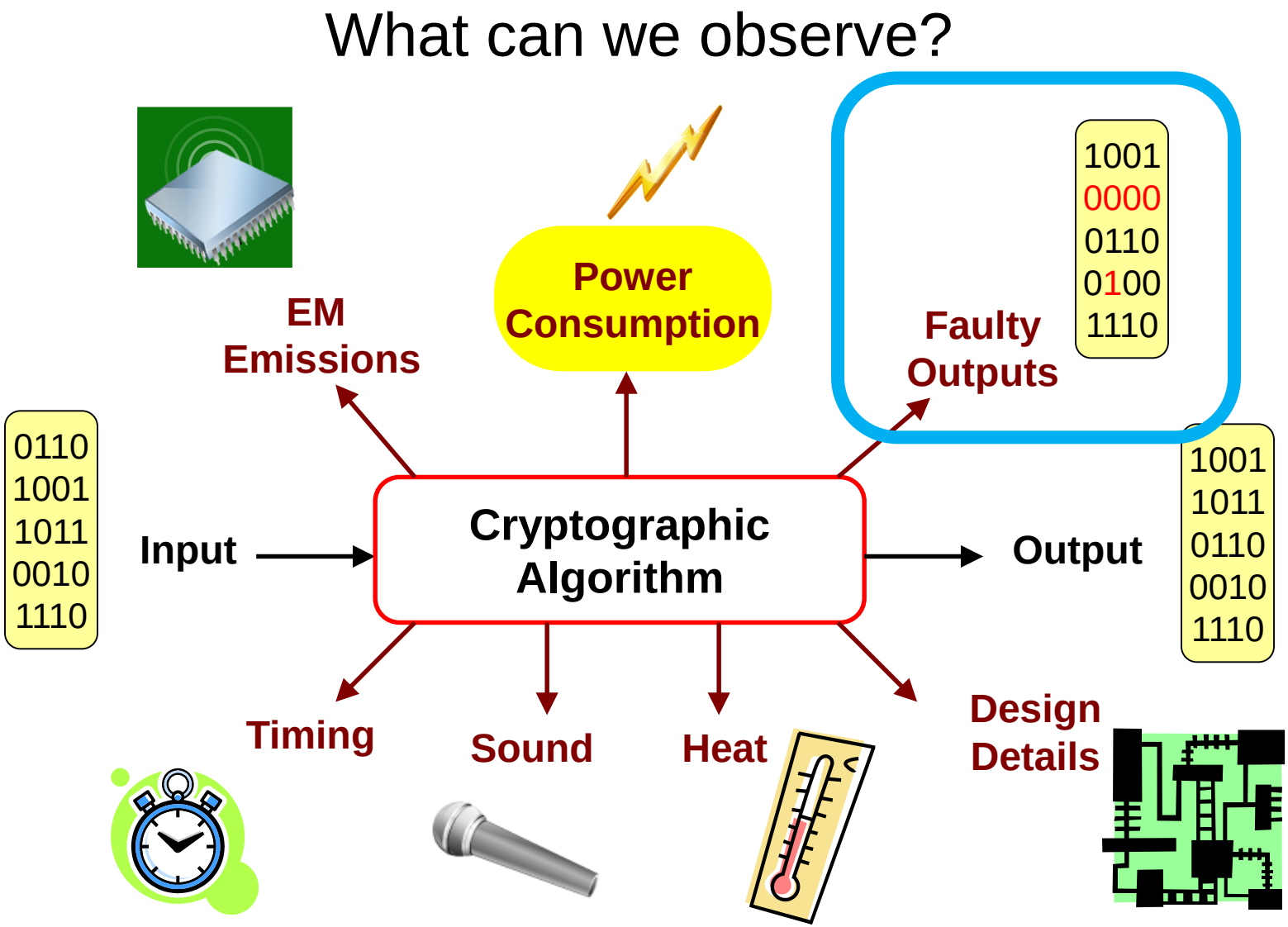
- Plaintext is encrypted using the Secret Key stored on chip.
- System will become useless if the adversary knows the Secret Key.

Side-Channel Attack




Prof. Patrick Schaumont's Lab
ECE Dept., Virginia Tech

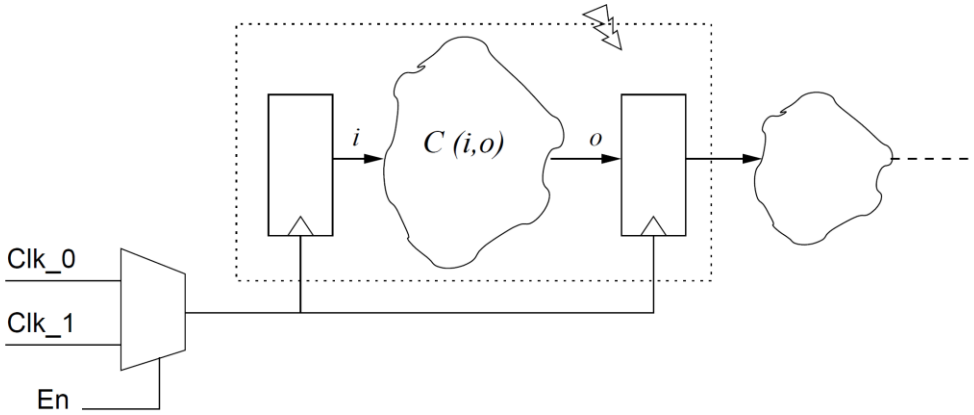
Side Channels

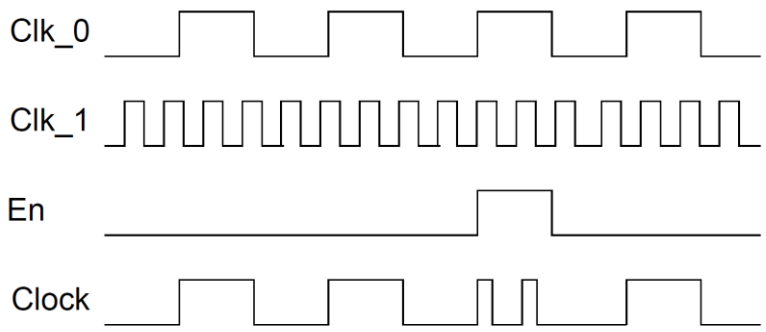


Fault Sensitivity Attack

[Ghalaty et al 2014, 2015]

- The goal of fault injection is to induce sufficiently many faulty outputs to reveal the secret key.





Fault Sensitivity Attack

[Ghalaty et al 2014, 2015]

- Exploiting dependence between ***secret key*** and the circuit's ***fault sensitivity***

Our Vision

Security by Compilation

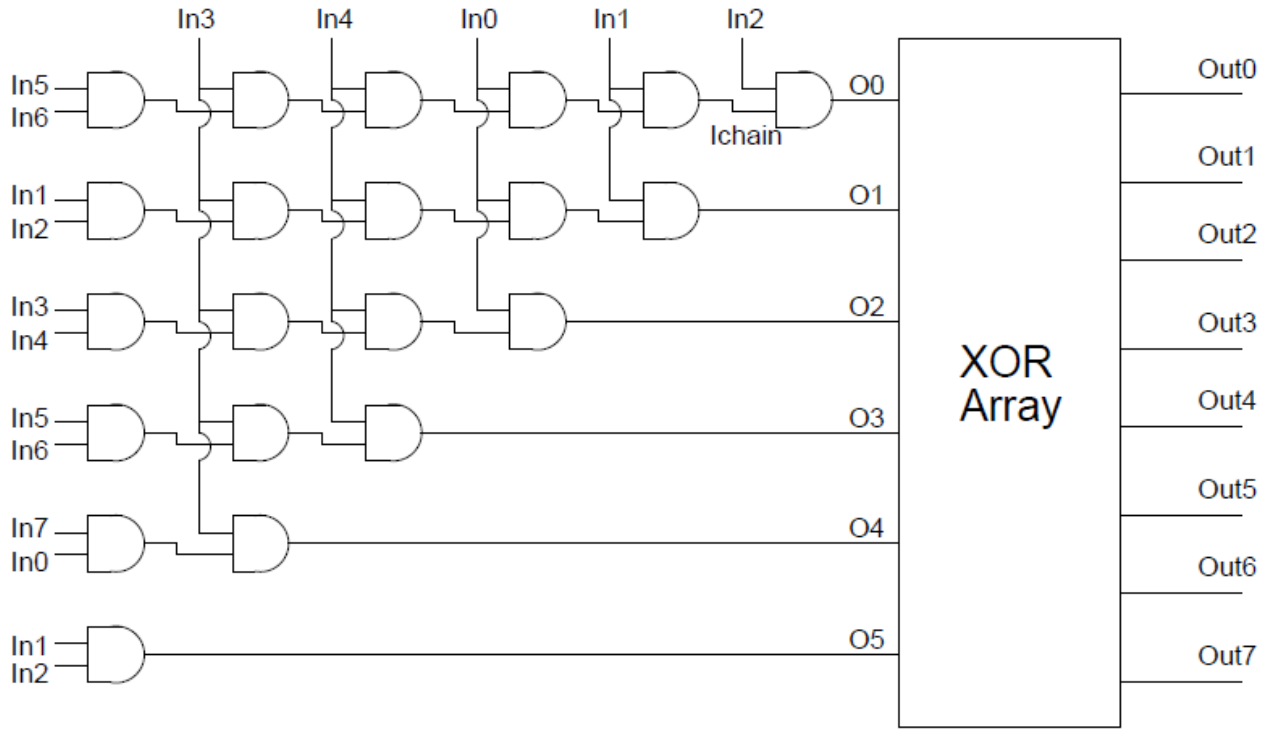
Unprotected HW / SW code

Protected HW / SW code

TWC: Small: Secure by Compilation: An Automated Approach to Comprehensive Side Channel Resistance

PIs: Chao Wang, Patrick Schaumont

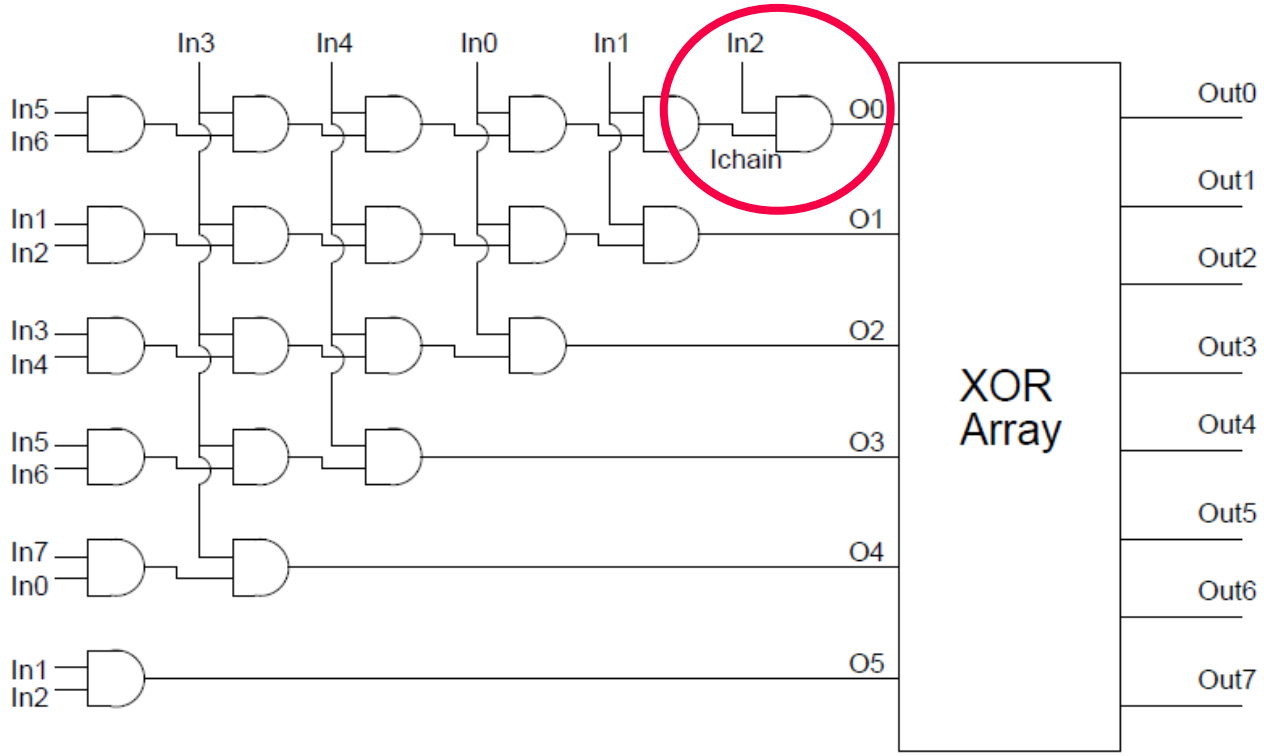
Motivating Example



PPRM1 AES S-box implementation [Morioka & Satoh, in CHES 2002]

1. *The only non-linear function in AES*
2. *Vulnerable to FSA attack*

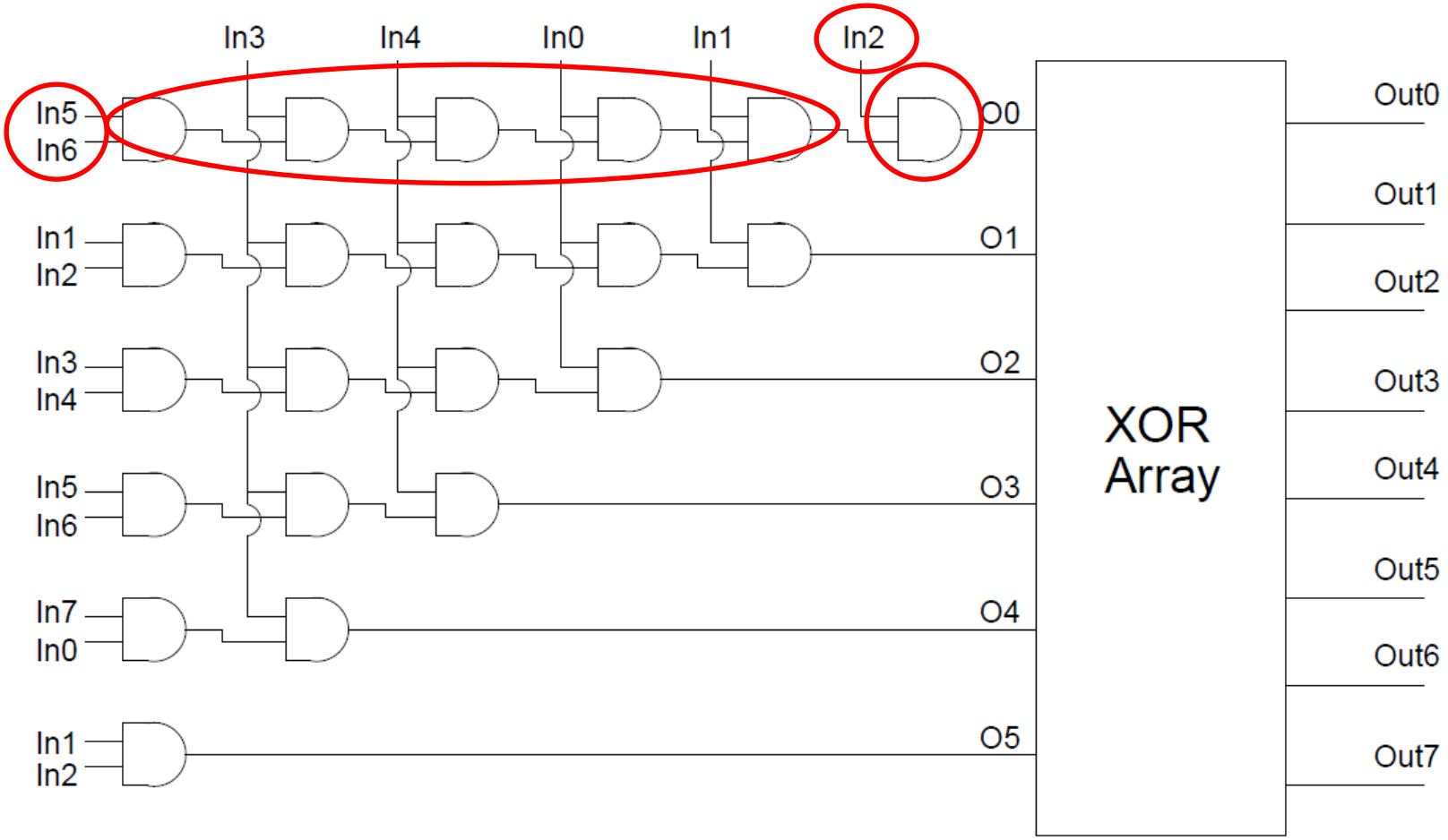
Motivating Example



PPRM1 AES S-box implementation *[Morioka & Satoh, in CHES 2002]*

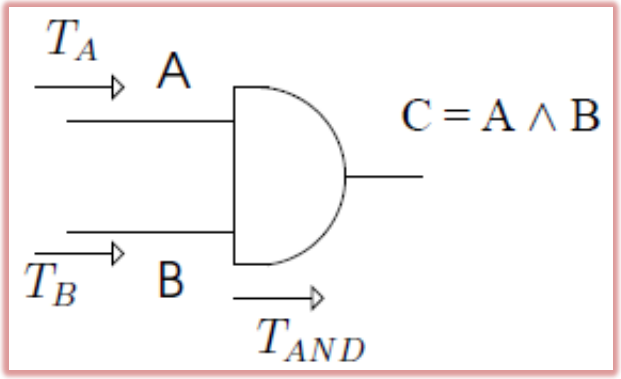
1. *The only non-linear function in AES*
2. *Vulnerable to FSA attack*

Motivating Example



Signal Delay for AND Gates

T_A -- arrival time for signal A
T_B -- arrival time for signal B
T_C -- arrival time for signal C
T_{AND} -- delay of AND gate



Assume $T_A < T_B$

When signal A = 0, $T_C = T_A + T_{AND}$ (small)
When signal A = 1, $T_C = T_B + T_{AND}$ (large)

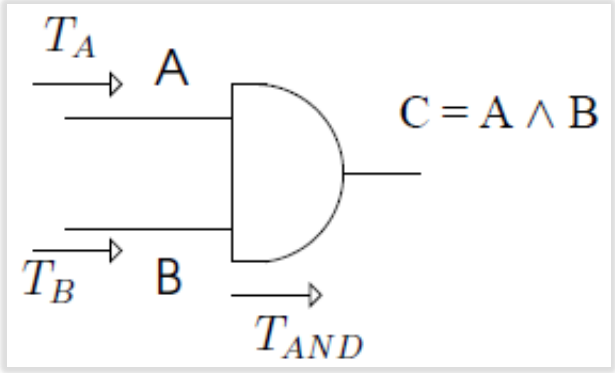
Timing Dependency !

Analyzing the (TC) may help decide the value of (A)

Signal Delay for AND Gates

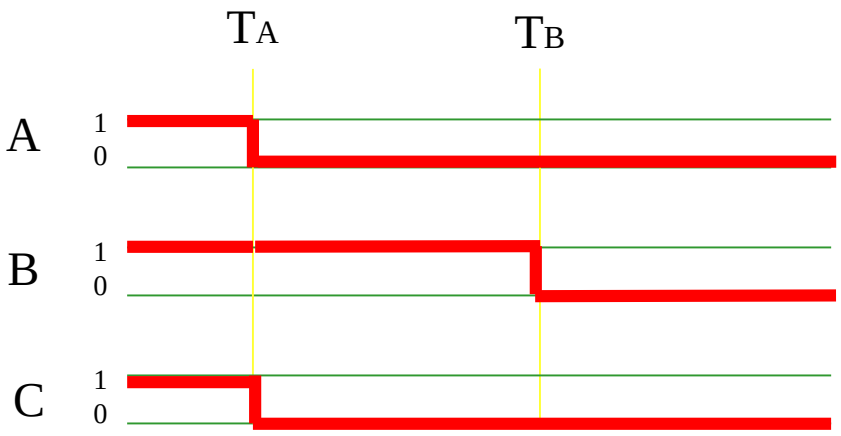
When signal A = 0, $T_C = T_A + T_{AND}$ (small)

When signal A = 1, $T_C = T_B + T_{AND}$ (large)



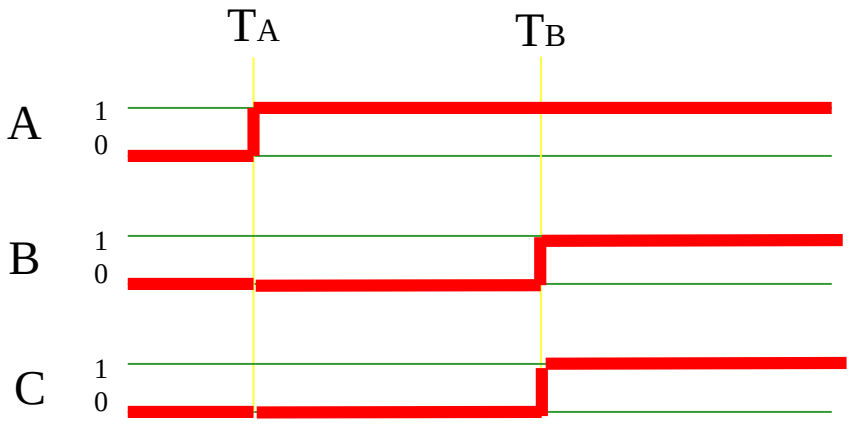
- If A = ’0’

  $T_C = T_A + T_{AND}$



- If A = ’1’

  $T_C = T_B + T_{AND}$



Countermeasure

- FSA Attack can be prevented by eliminating the timing dependency
  - Between signal path delay and sensitive input

Countermeasure Synthesis

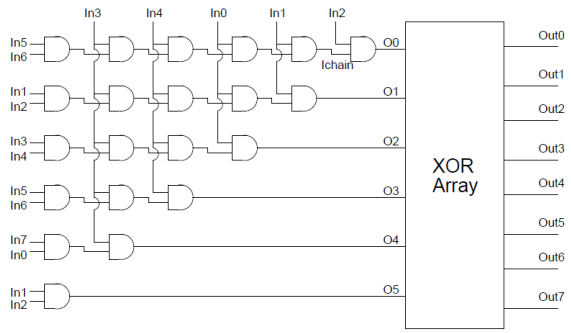


Fig. 1. PPRM1 AES S-box that is vulnerable to FSA.

Original circuit

[Morioka & Satoh, CHES 2002]

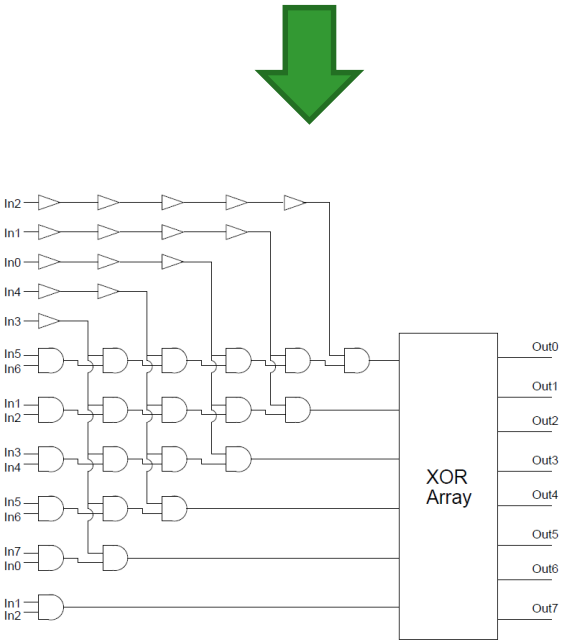


Fig. 2. S-box with buffered countermeasure [22].

Buffered circuit

[Ghalaty et al, DATE 2014]

[Endo et al, IEEE TVLSI, 2014]

Synthesized countermeasure

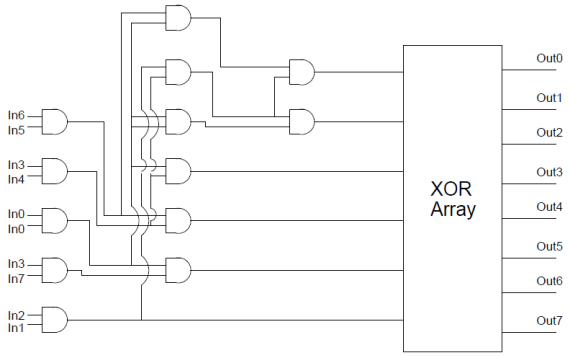
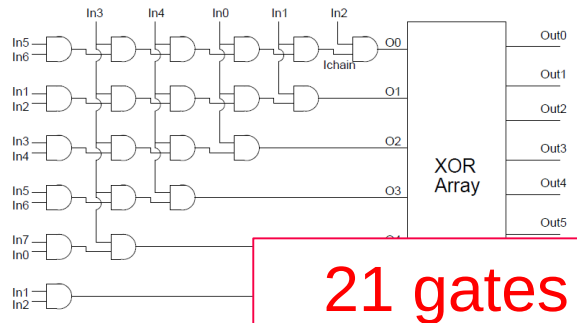


Fig. 3. S-box with our new countermeasure.

Advantage

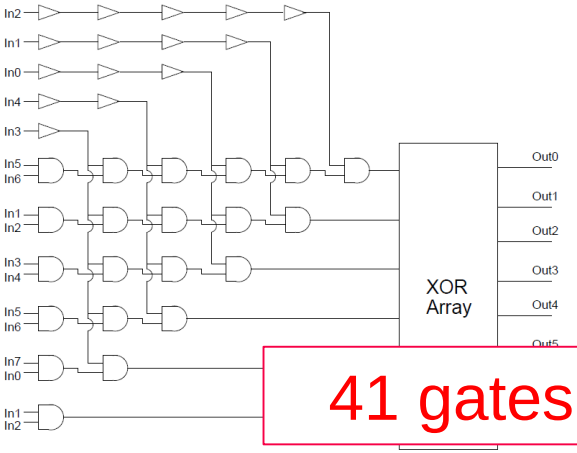
Smaller Circuit



21 gates

Original circuit
[Morioka & Satoh, CHES 2002]

Fig. 1. PPRM1 AES S-box that is vulnerable to FSA.

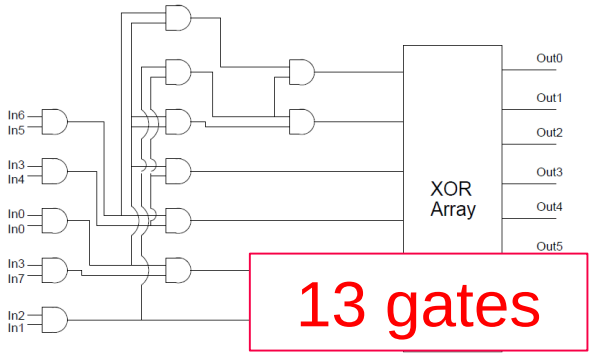


41 gates

Buffered circuit
[Ghalaty et al, DATE 2014]
[Endo et al, IEEE TVLSI, 2014]

Fig. 2. S-box with buffered countermeasure [22].

Synthesized countermeasure

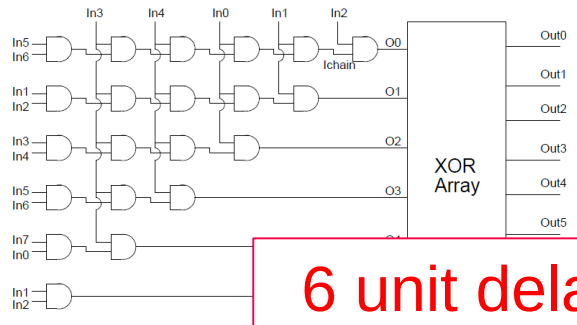


13 gates

Fig. 3. S-box with our new countermeasure.

Advantage

Shorter Critical Path

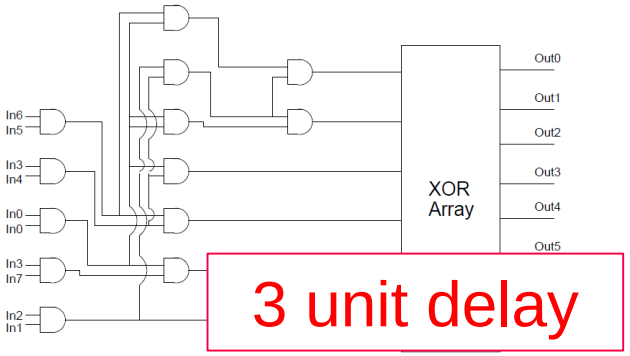


Original circuit
[Morioka & Satoh, CHES 2002]

6 unit delay

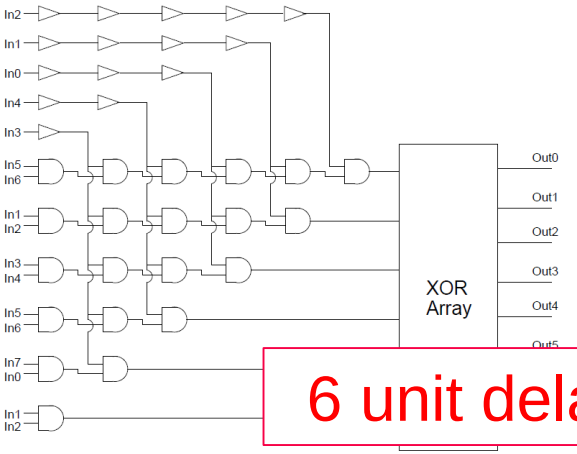
Fig. 1. PPRM1 AES S-box that is vulnerable to FSA.

Synthesized countermeasure



3 unit delay

Fig. 3. S-box with our new countermeasure.



Buffered circuit
[Ghalaty et al, DATE 2014]
[Endo et al, IEEE TVLSI, 2014]

6 unit delay

Fig. 2. S-box with buffered countermeasure [22].

Advantage

Cooler Technology

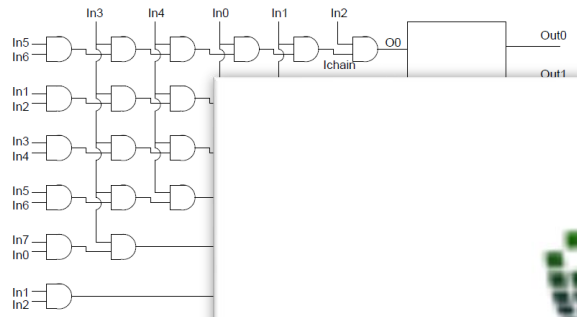


Fig. 1. PPRM1 AE

ynthesized
ountermeasure

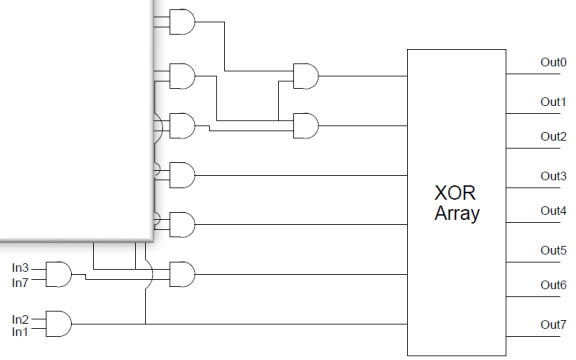


Fig. 3. S-box with our new countermeasure.

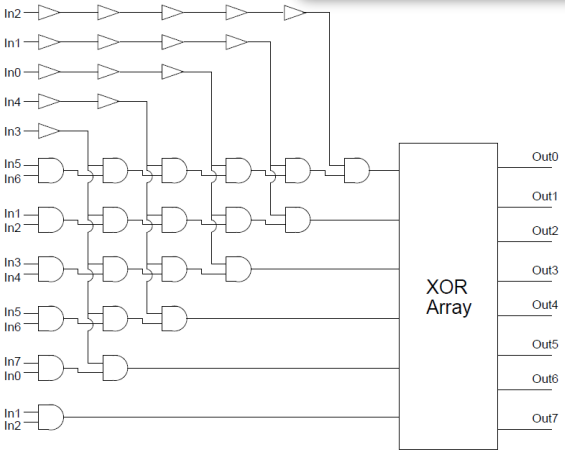


Fig. 2. S-box with buffered countermeasure [22].

Buffered circuit

[Ghalaty et al, DATE 2014]

[Endo et al, IEEE TVLSI, 2014]

Advantage

Cooler Technology

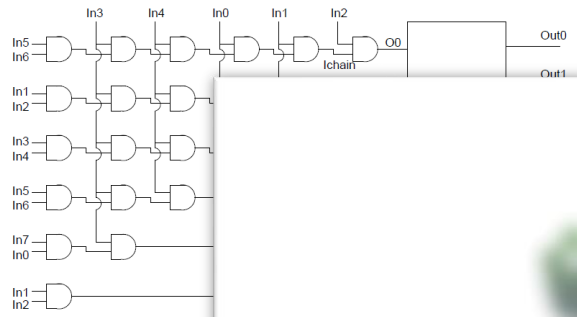


Fig. 1. PPRM1 AE

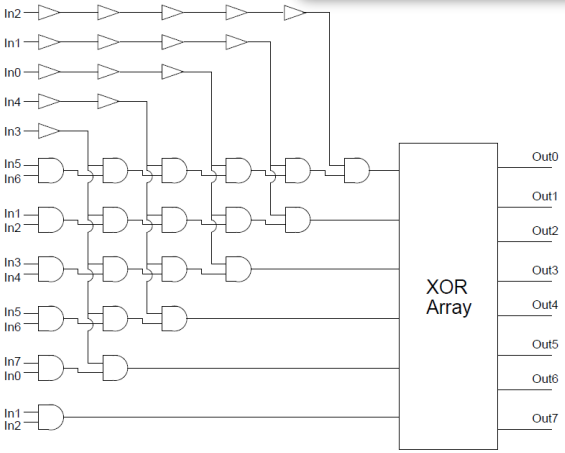


Fig. 2. S-box with buffered countermeasure [22].

Buffered circuit
[Ghalaty et al, DATE 2014]
[Endo et al, IEEE TVLSI, 2014]

ynthesized
ountermeasure

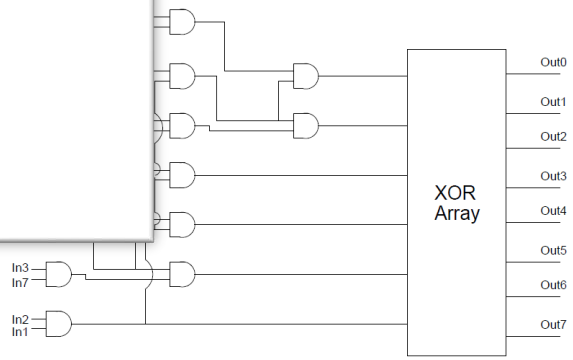
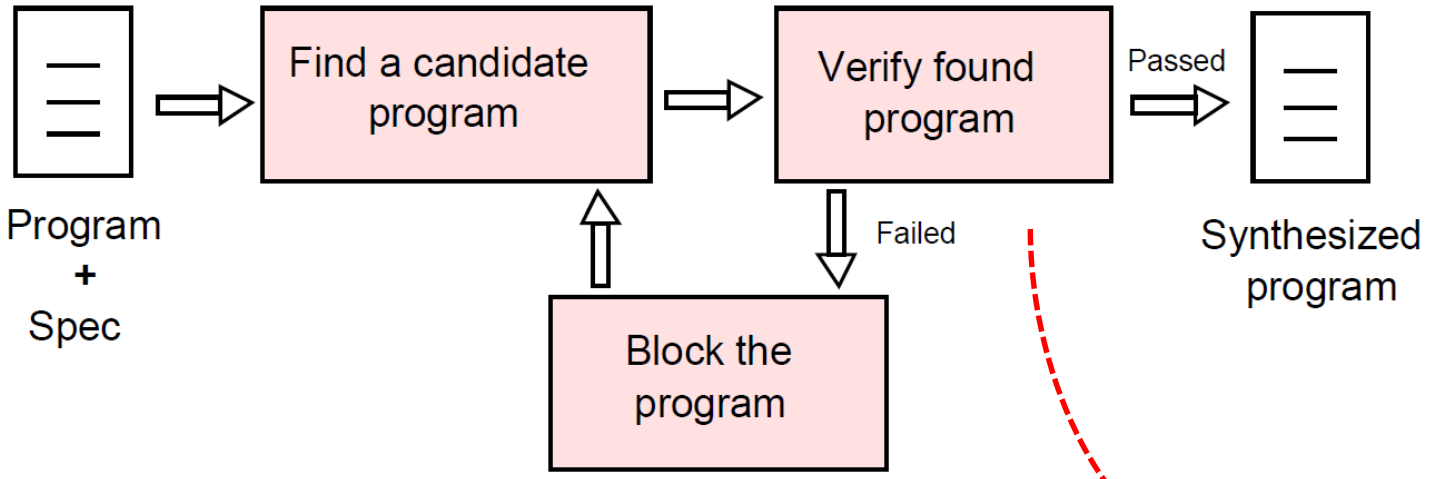


Fig. 3. S-box with our new countermeasure.

Our Contribution

- The first countermeasure synthesis method to defend against FSA attacks of crypto circuits

Inductive Synthesis

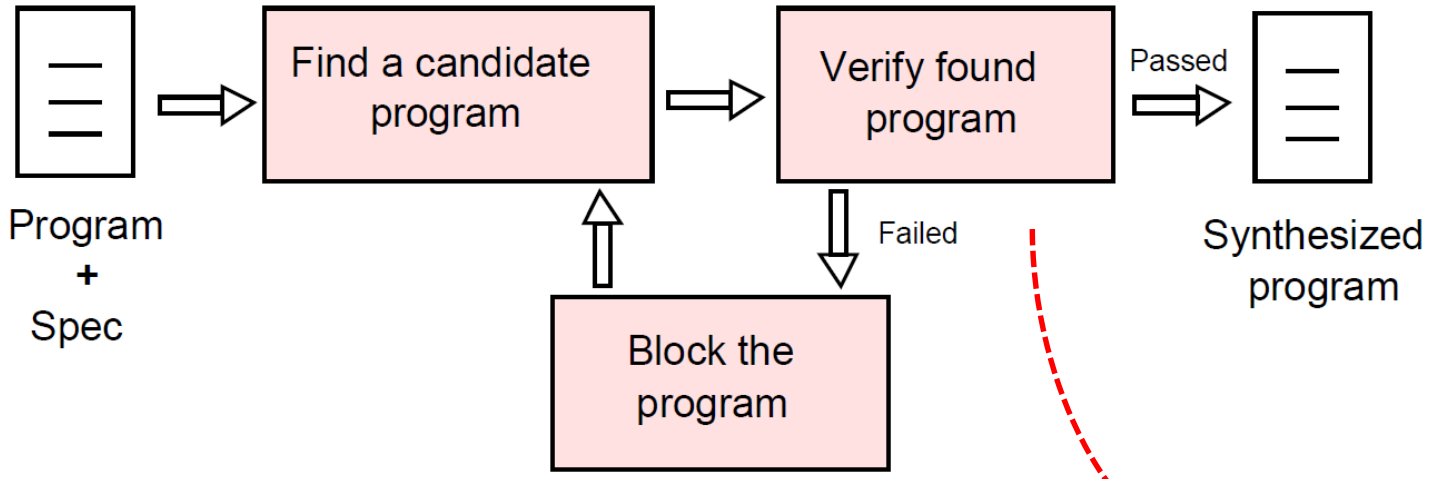


Verification:

(1) Checking the functional equivalence

(2) Checking the FSA resistance

Inductive Synthesis



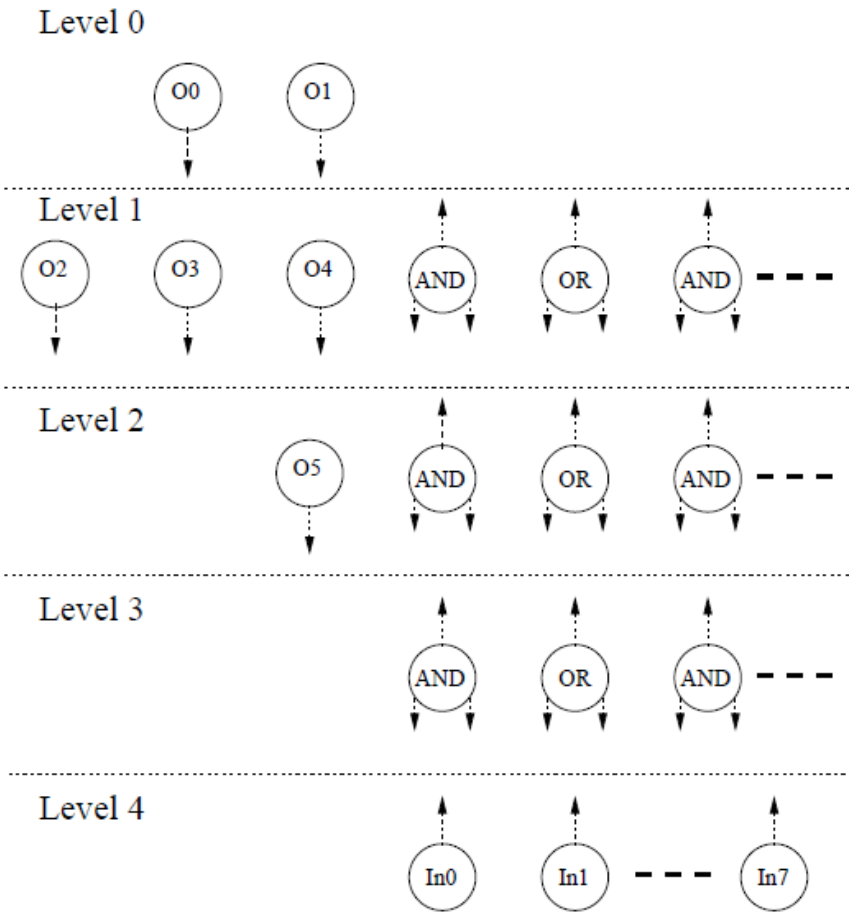
Verification:

(1) Checking the functional equivalence

~~(2) Checking the FSA resistance~~

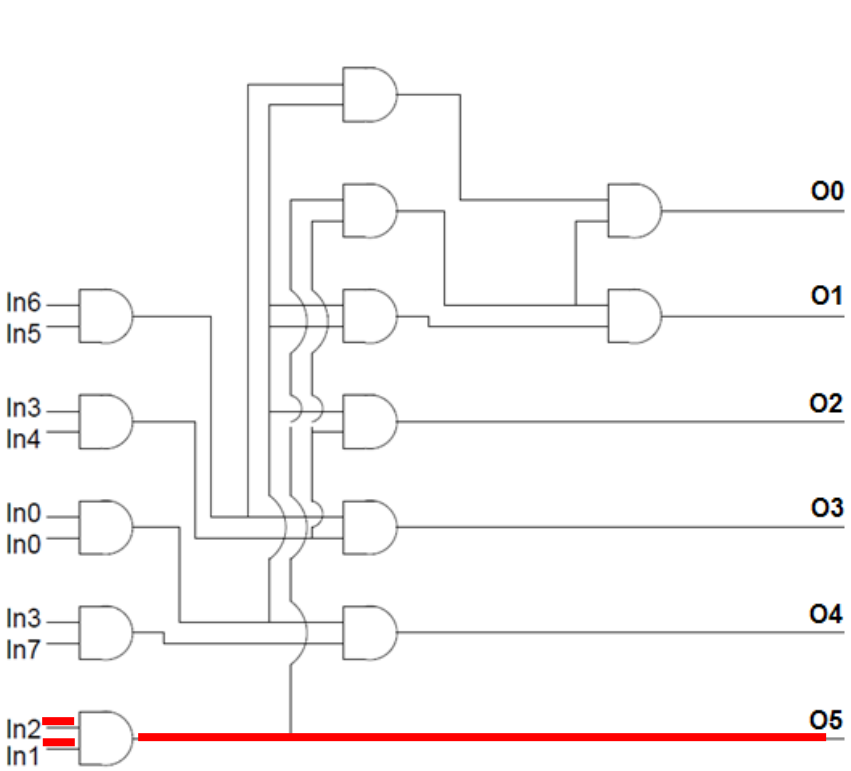
Template Circuit

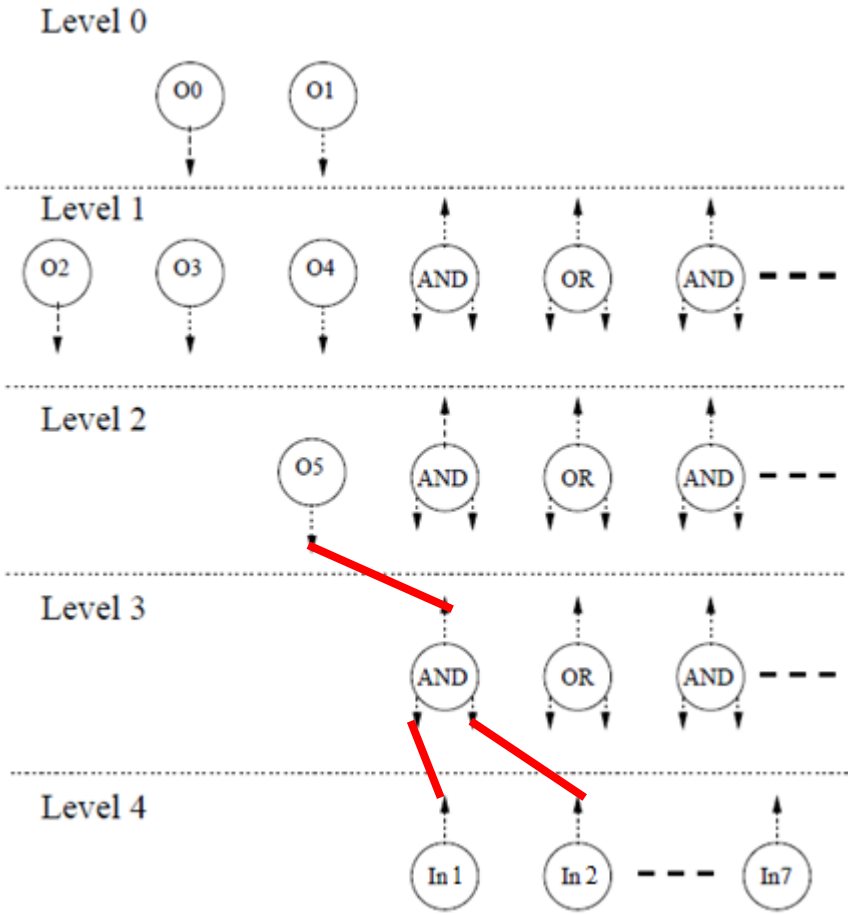
- FSA resistance by construction



Template Circuit

- FSA resistance by construction





Template Circuit

- SyGuS specification to generate instantiation (candidate circuit)

```
(define-fun Spec ((i0 Bool)(i1 Bool)(i2 Bool)(i3 Bool)(i4 Bool)(i5 Bool)(i6
    Bool)) Int
  (+ (ite (and i2 (and i1 (and i0 (and i4 (and i3 (and i5 i6)))))) 1 0 )
     (ite (and i1 (and i0 (and i4 (and i3 (and i1 i2))))) 2 0 ) )
)
(synth-fun Impl ((i0 Bool)(i1 Bool)(i2 Bool)(i3 Bool)(i4 Bool)(i5 Bool)(i6
    Bool)) Int
  ((Start Bool ( (+ (ite d0 1 0)
                    (ite d0 2 0)) ) )
   (d0 Bool ( (and d1 d1)
              (or d1 d1) ) )
   (d1 Bool ( (and d2 d2)
              (or d2 d2) ) )
   (d2 Bool ( (and d3 d3)
              (or d3 d3) ) )
   (d3 Bool ( i0 i1 i2 i3 i4 i5 i6 ) ) )
)
(constraint (= (Spec i0 i1 i2 i3 i4 i5 i6) (Impl i0 i1 i2 i3 i4 i5 i6) ) )
```

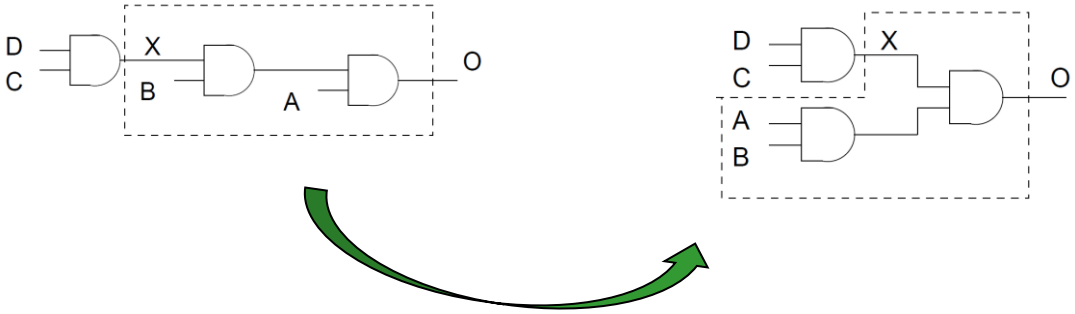
Scalability Problem

- Our solution: Partitioned Synthesis
  - (1) Divide the circuit into smaller regions
  - (2) Synthesize countermeasures for each region
  - (3) Compose them together

Compositionality:

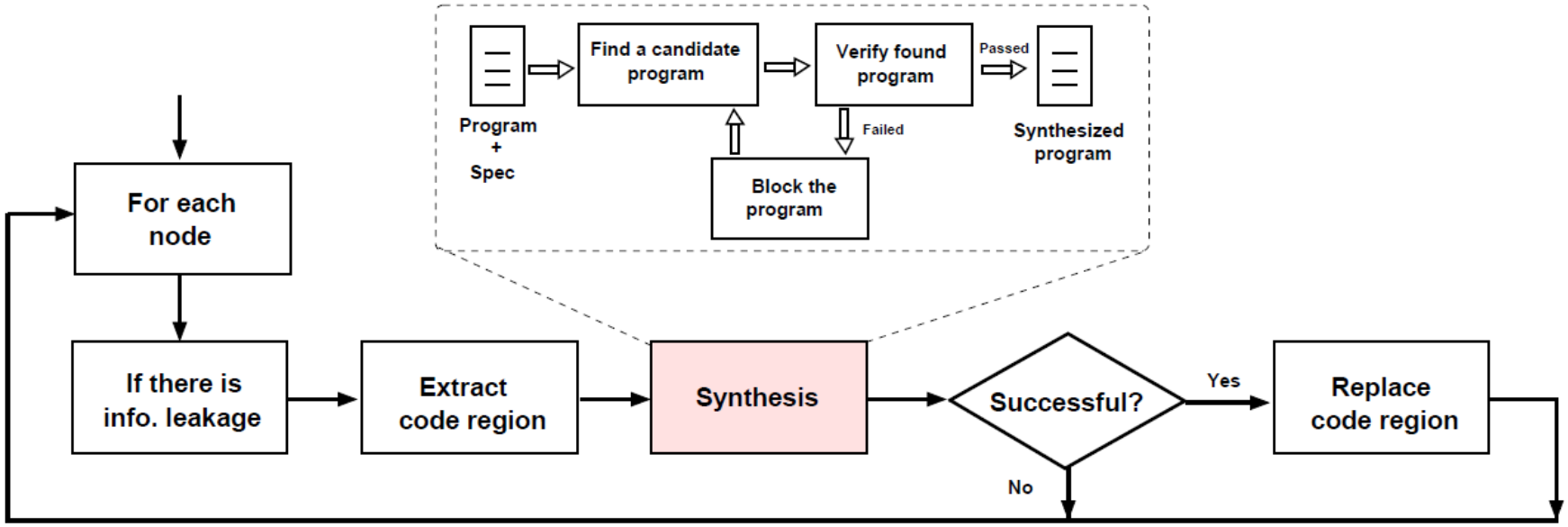
(1) The delay of a path is the summation of delays of all segments
(2) If each region is FSA resistant, the entire circuit is FSA resistant

Partitioned Synthesis



Algorithm 1 Partitioned FSA-countermeasure synthesis procedure.

```
1:  GEN-COUNTERMEASURE (P, InputSort, GatesPD, GatesSyn, lev) {
2:    while (true) {
3:      for each (gate g ∈ P) {
4:        MaxPD[g] ← GETMAXPD (g, GatesPD, P)
5:        MinAr[g] ← GETMINAR (g, GatesPD, P)
6:        MaxAr[g] ← GETMAXAR (g, GatesPD, P)
7:      }
8:      sGate ← GETSENSITIVE (MaxPD, MinAr, MaxAr, P)
9:      if (sGate = ∅)
10:       return P;
11:     n ← 2^lev − 1
12:     newReg ← ∅
13:     while (newReg = ∅) {
14:       reg ← GETREG (sGate, MinAr, MaxAr, P, n)
15:       newReg ← SYNTHESIZE (reg, MinAr, GatesSyn, lev)
16:       n ← n − 1
17:     }
18:     P ← UPDATEREGION (P, reg, newReg)
19:   }
20: }
```



Experiments

- Implemented in a software tool
  - Circuit-to-SyGuS translator
  - + SyGuS solvers (www.sygus.org)

- Evaluated on 10 crypto circuits

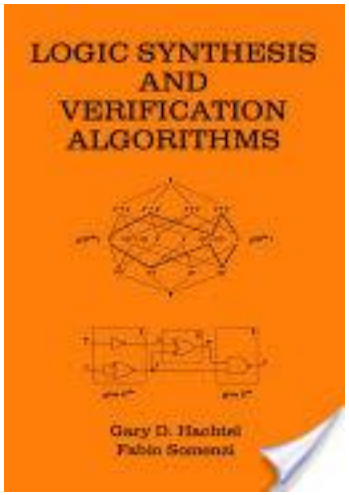
Name	Circuit Description	Nodes	Inputs	Outputs
C1	MAC-Keccak nonlinear masked Chi function 1 [8]	35	10	1
C2	MAC-Keccak nonlinear masked Chi function 2 [8]	35	10	1
C3	Generated MAC-Keccak nonlinear masked Chi function 1 [14]	44	10	1
C4	Generated MAC-Keccak nonlinear masked Chi function 2 [14]	44	10	1
C5	Unmasked MAC-Keccak nonlinear Chi function [8]	6	3	1
C6	AES S-Box design of nonlinear invg4 function [11]	83	4	4
C7	AES S-Box design of nonlinear mul4 function [11]	63	8	4
C8	AES S-Box single round nonlinear functions [11]	209	8	8
C9	Complete AES PPRM1 S-box design [34]	8,054	8	8
C10	Complete AES Boyar-Peralta S-box design [11]	156	8	8

Compared to Buffer Insertion Methods

Name	Nodes	Buffer Insertion		New Method	
		nodes	increase	nodes	increase
C1	35	51	45%	42	20%
C2	35	48	37%	40	14%
C3	44	54	22%	48	9%
C4	44	59	34%	45	2%
C5	6	9	50%	9	50%
C6	83	134	61%	98	18%
C7	63	79	25%	73	15%
C8	209	292	39%	244	16%
C9	8,054	77,717	864%	8,943	11%
C10	156	9,585	6044%	370	137%

Existing countermeasures (buffer insertion)
[Ghalaty et al, DATE 2014]
[Endo et al, IEEE TVLSI, 2014]

Compared to Classic EDA Algorithms



Logic synthesis and optimization algorithms

- Two-Level Minimization
- Multi-Level Minimization
- …

ABC

A System for Sequential Synthesis and Verification

Berkeley Logic Synthesis and Verification Group

Comparing our method with the *balance* command of ABC

Compared to Classic EDA Algorithms

Comparing our method with the *balance* command of ABC

Name	**Depth**	**ABC**		**Our New Method**	
		Node Increase	Depth	Node Increase	Depth
C1	8	300%	27	20%	5
C2	7	300%	27	31%	6
C3	7	273%	25	20%	6
C4	8	273%	28	18%	6
C5	3	233%	7	50%	3
C6	9	285%	31	18%	7
C7	7	322%	21	16%	7
C8	17	308%	33	17%	15
C9	156	80%	586	11%	17
C10	24	476%	64	137%	23

Compared to Classic EDA Algorithms

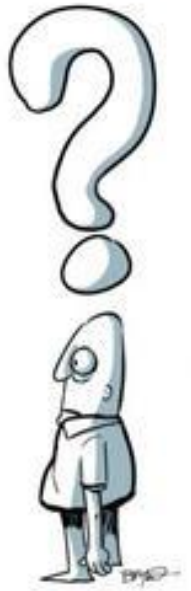
Comparing our method with the *balance* command of ABC

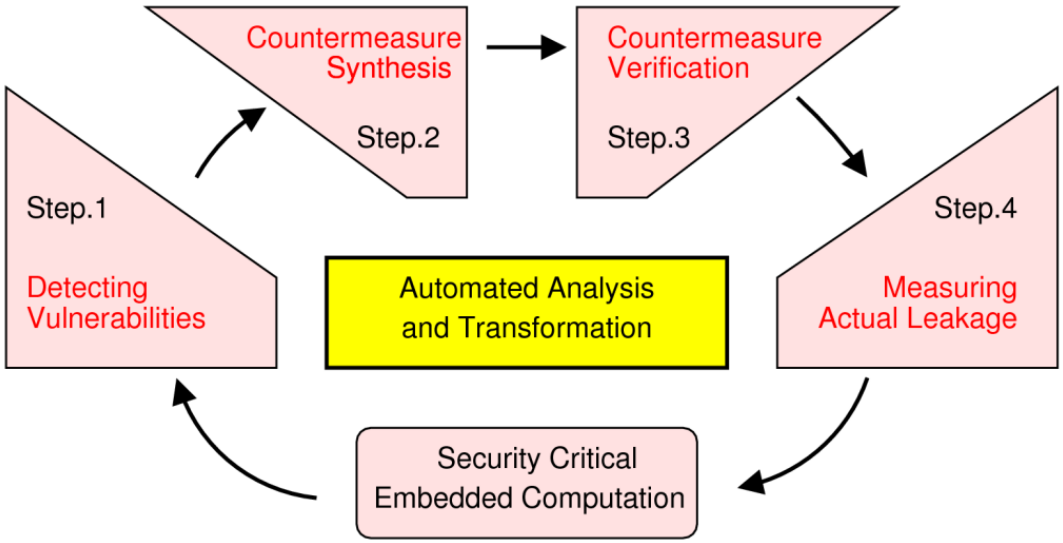
Name	**Depth**	**ABC**		**Our New Method**	
		Node Increase	Depth	Node Increase	Depth
C1	8	300%	27	20%	5
C2	7	300%	27	31%	6
C3	7	273%	25	20%	6
C4	8	273%	28	18%	6
C5	3	233%	7	50%	3
C6	9	285%	31	18%	7
C7	7	322%	21	16%	7
C8	17	308%	33	17%	15
C9	156	80%	586	11%	17
C10	24	476%	64	137%	23

Conclusions

- New countermeasure synthesis method for FSA attacks of crypto circuits
  - Guarantee to eliminate sensitive timing dependency
  - Efficient (Fewer gates, Shorter critical paths, etc.)
- Future work
  - Synthesizing countermeasures for other side-channel attacks

Security by Compilation





[TACAS14]
[CAV14]
[DAC14]
…
[CAV16]