# Logic Locking: Current Trends, Attacks and Future Directions

**Ujjwal Guin**, Assistant Professor
Electrical and Computer Engineering
Auburn University, Auburn, AL 36849
ujjwal.guin@auburn.edu

**Pramod Subramanyan**, Assistant Professor
Computer Science and Engineering
Indian Institute of Technology, Kanpur
spramod@cse.iitk.ac.in

# Part I: Motivation and Introduction to Logic Locking

# Outline

- IC Counterfeiting and Security Concerns
- Design-for-Security (DFS) Requirements
- Initial Proposals for Logic Locking
- Threat and Adversary Modeling
- Attacks on Logic Locking
- Countermeasures
- Future Directions
- Practical Session: Encryption Tools, Attack Tools, etc.

# Problem: Counterfeiting and Piracy

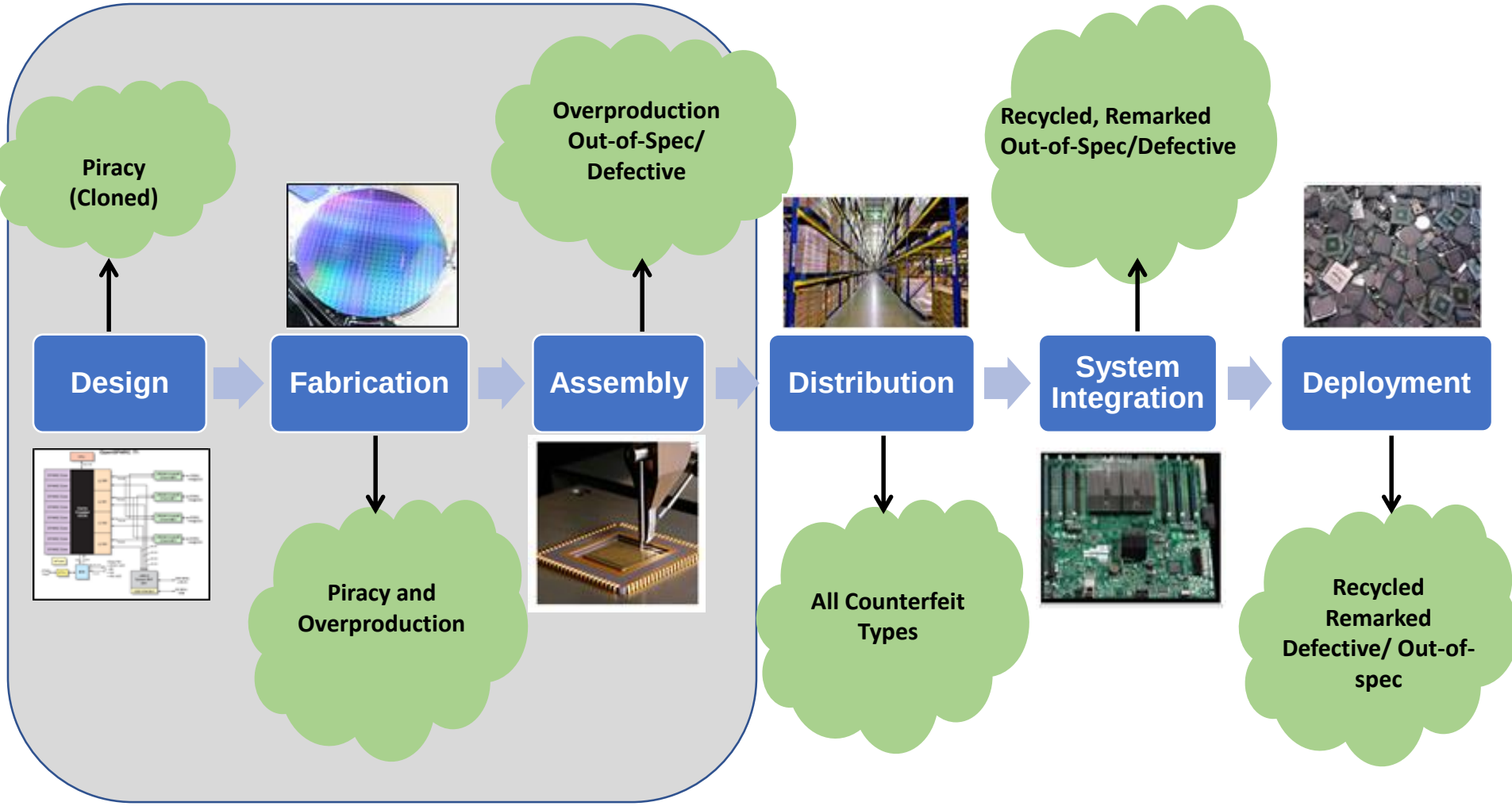
- Counterfeiting and Piracy are one of the emerging and evolving threat.

- Amount of total counterfeiting globally has reached to 1.2 Trillion USD in 2017 and is bound to reach 1.82 Trillion USD by the year 2020 [Business Wire].

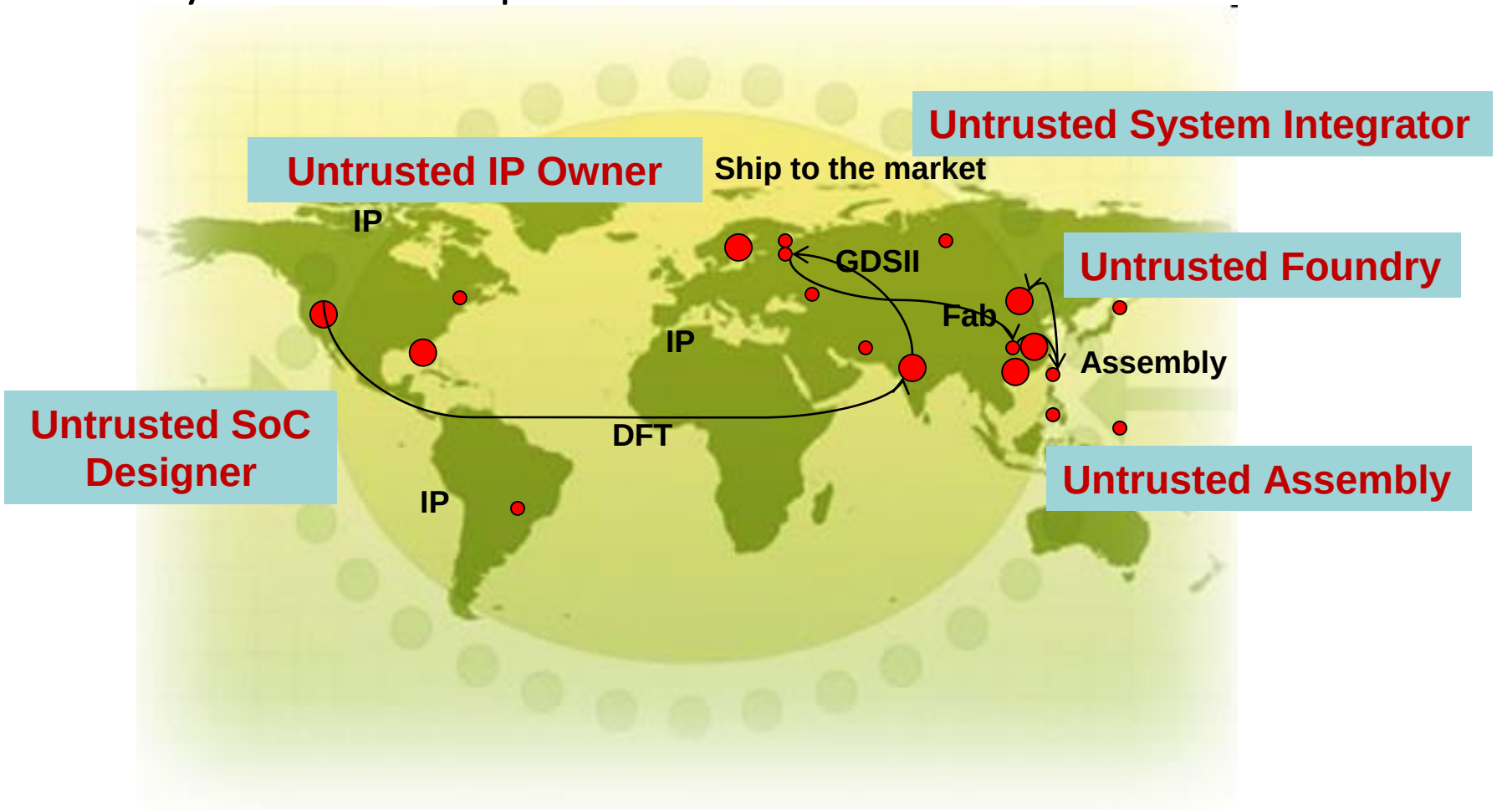
- Total cost of counterfeiting and piracy for G20 nations was $450 to $650 billion in 2008 and will grow to $1.2 to1.7 trillion in 2015 [ICC].

- Counterfeit parts pose a risk of $169B per year for global electronic supply chain [IHS].

# Supply Chain Vulnerabilities



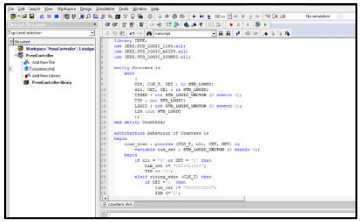
# Piracy and Overproduction



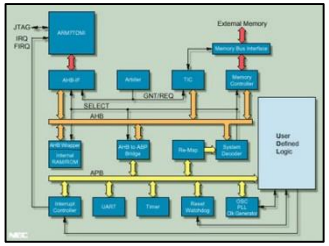
# Piracy and Overproduction- Cont.

- IP Overuse
  - An untrusted system on chip (SoC) designer produce more ICs and report a lesser number to the IP owner.
  - He may illegally use an IP that was licensed to be used in a different design.
- IC Overproduction
  - Untrusted foundries and assemblies may produce more than the number of chips they are contracted to manufacture.
- IP Piracy and Cloning
  - An untrusted SoC designer may legally purchase a 3PIP core from an IP vendor and then make clones, or illegitimate copies of the original IP.
  - An untrusted SoC designer can add some extra features to those 3PIPs to make them look like a different one and then sell them to another SoC designer.
  - An SoC designer may also modify a 3PIP in order to introduce a backdoor or hardware Trojan into the chip.

# Problem Statement



IP Owners

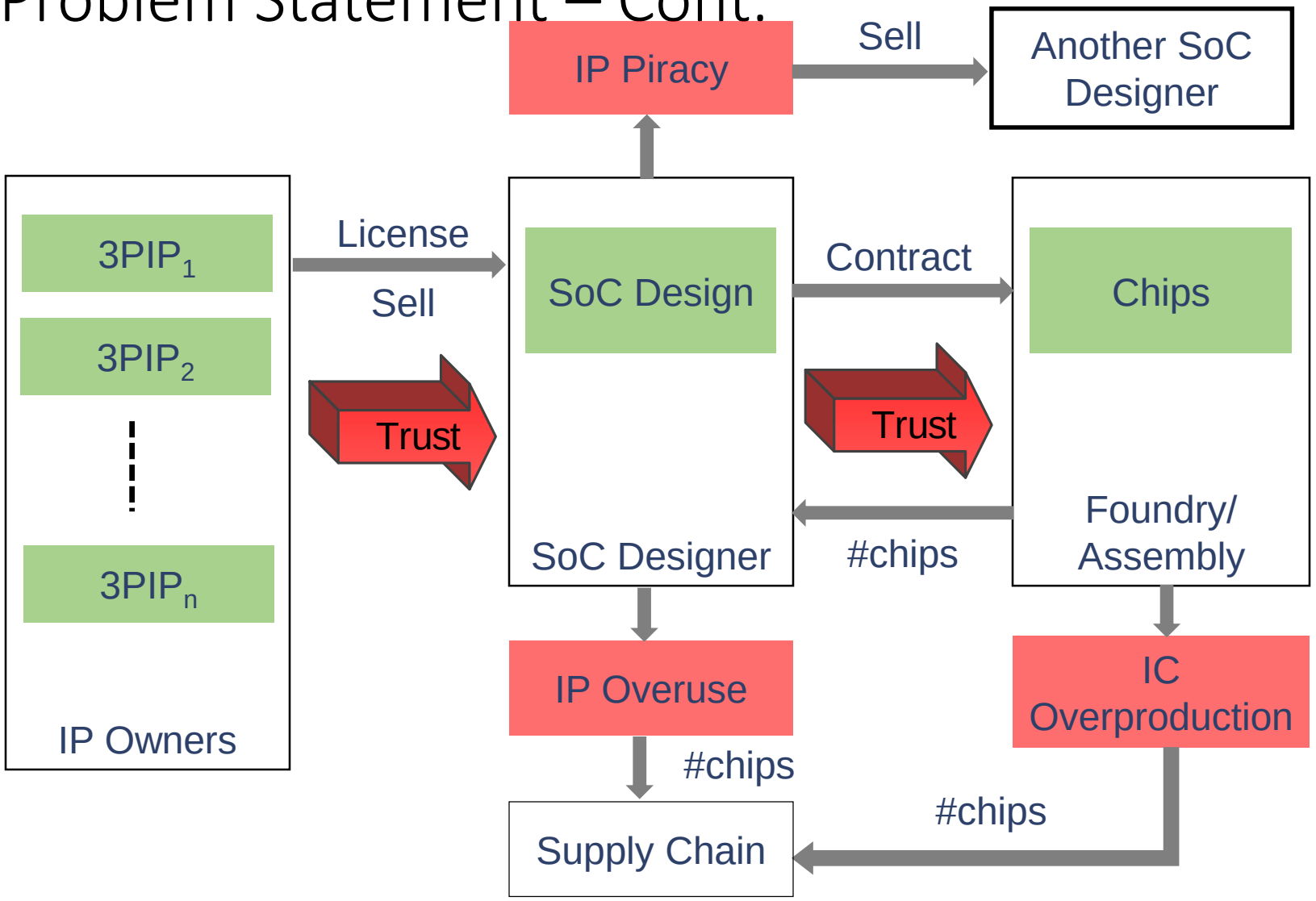


SoC Designers

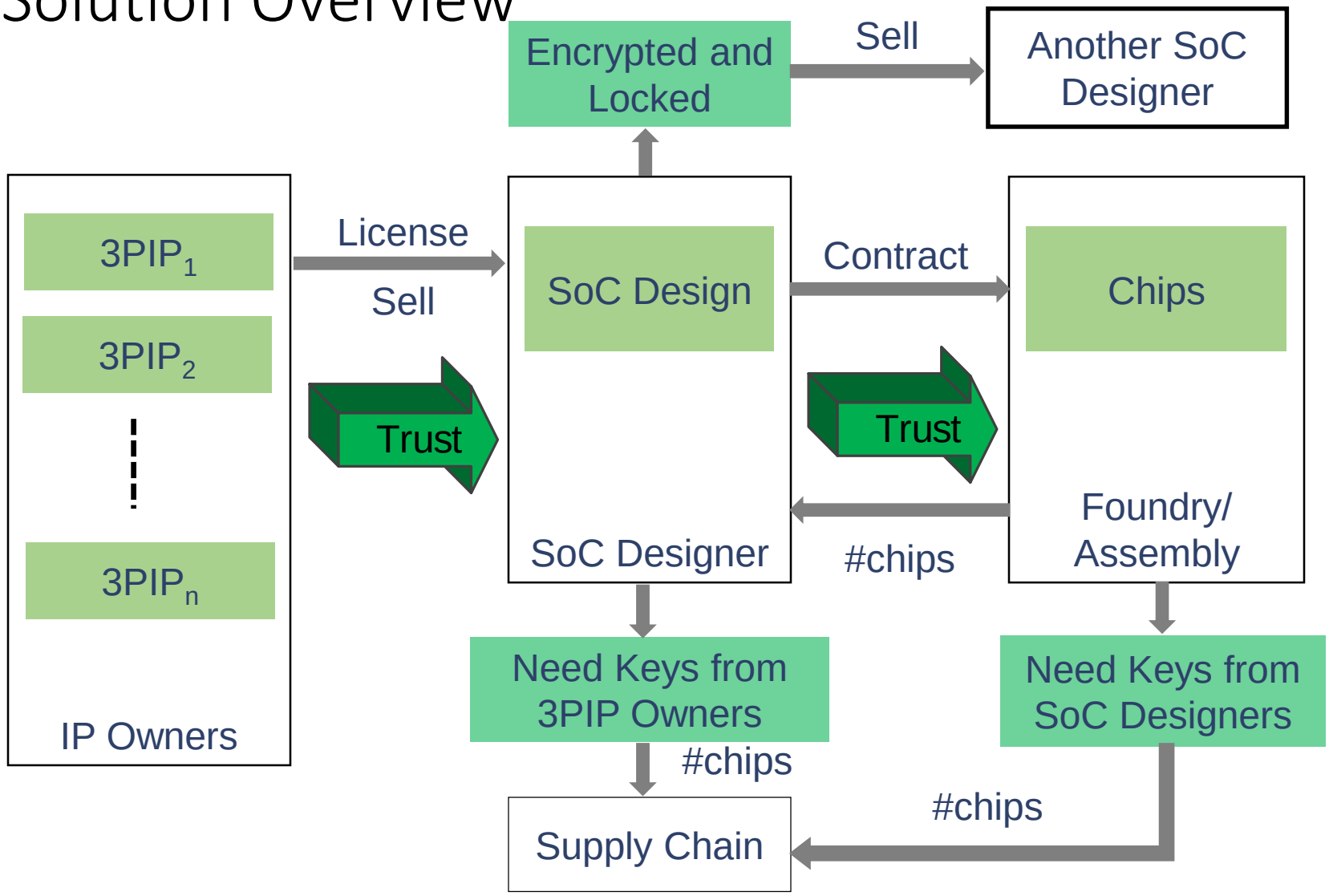
Foundry/
Assembly

Images: Google

# Problem Statement – Cont.



# Solution Overview



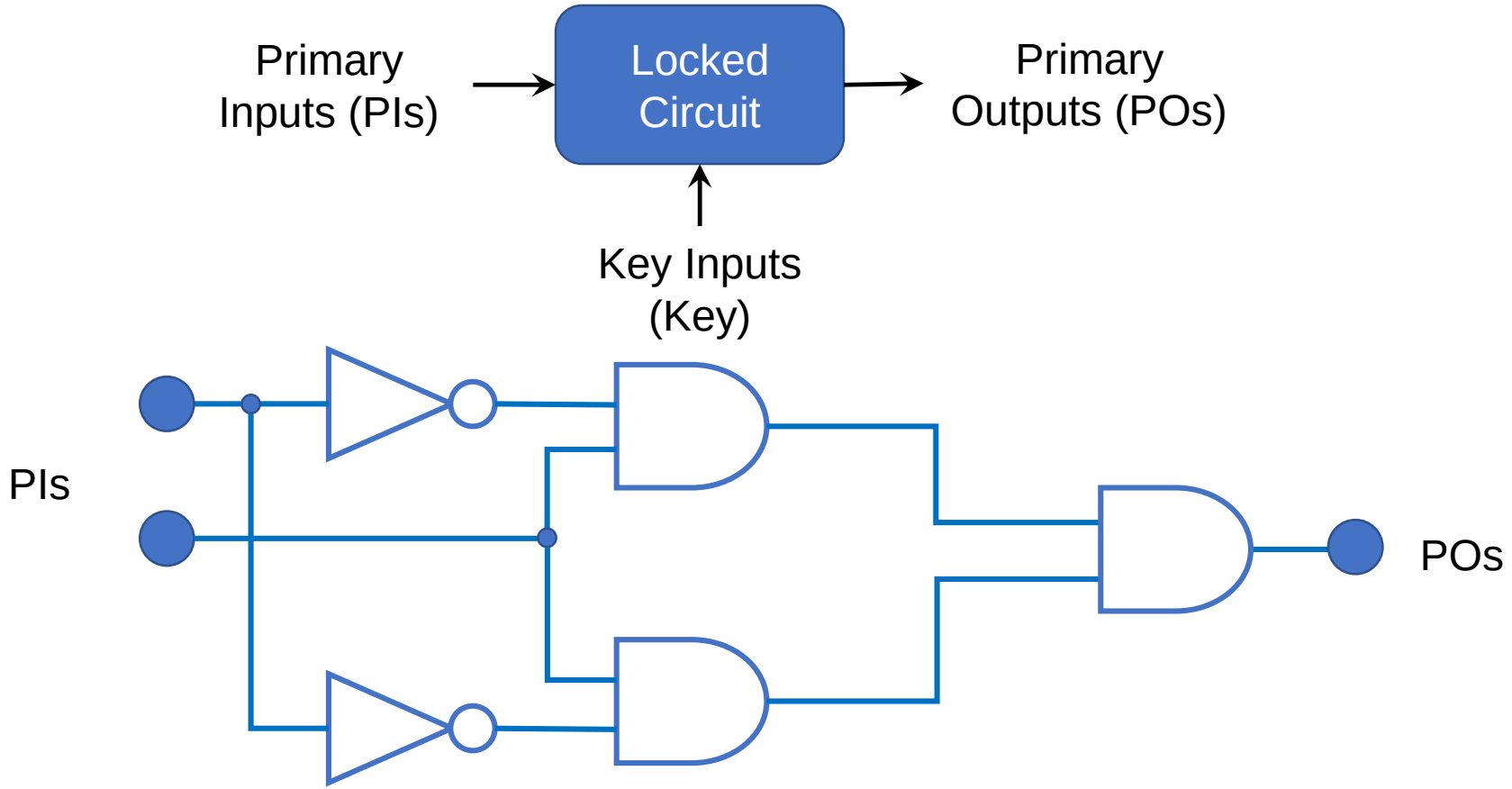
# Solution Overview- Cont.

- Hardware watermarking
  - Creating a unique fingerprint for verify proof of IP ownership
  - Cannot be used it to prevent IP overuse, IP piracy, and IC overproduction
- IC camouflaging
  - A layout-level technique to hamper image processing-based extraction of the gate-level netlist.
  - Difficulty of implementing metering.

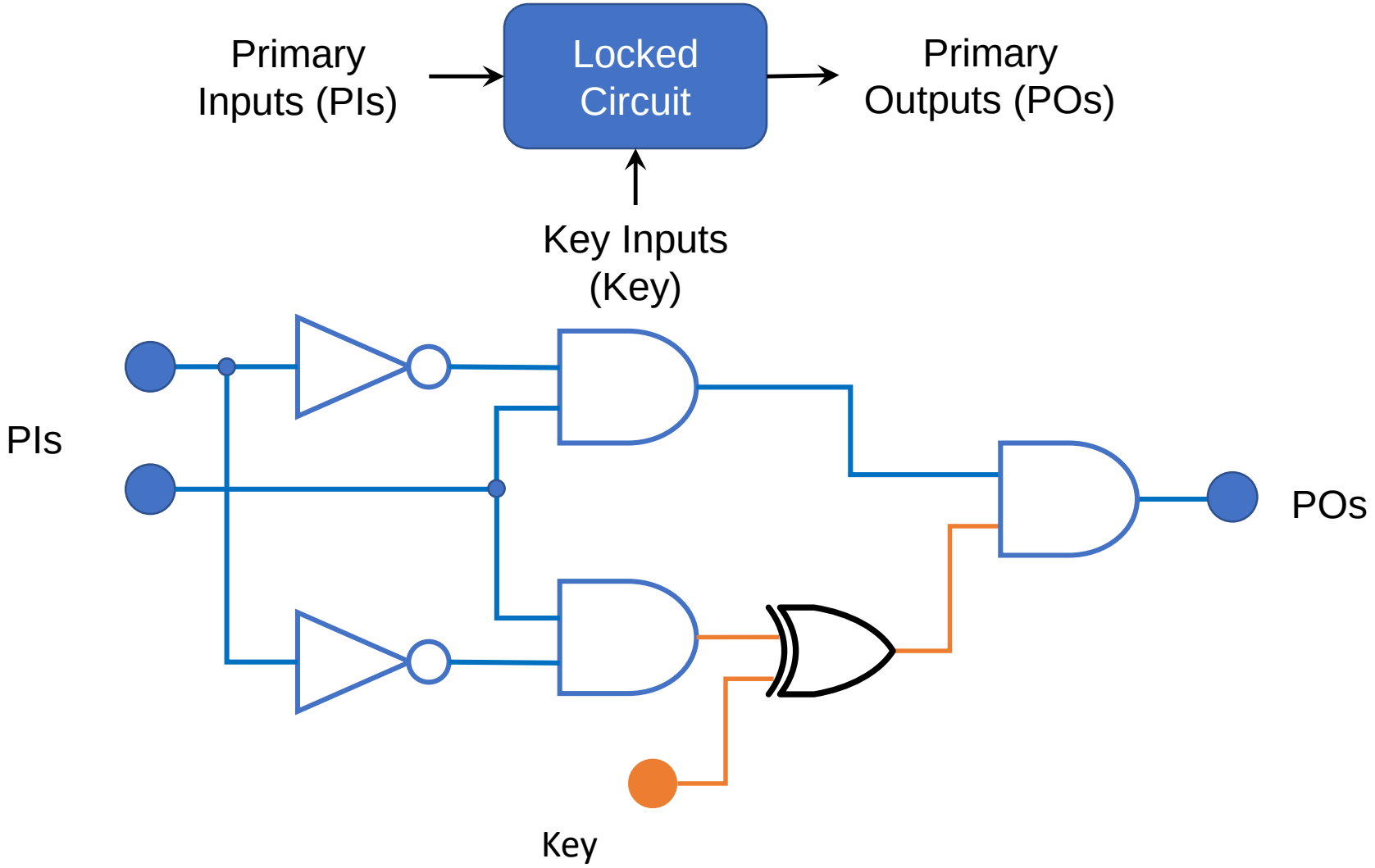
# Solution Overview- Cont.

- IC metering
  - Prevent IC overproduction by control the number of ICs  manufactured
  - Metering using logic locking can be used to prevent all the aforementioned attacks.
- Split manufacturing
  - The design is split into the different layers and fabricated separately in different foundries.
  - The cost will be higher than the conventional methods
- Logic Locking
  - Obfuscate the inner details of the original design
  - A secret key can only unlock the chips.
  - Should be resistant to different attacks.

# Initial Proposal for Logic Locking



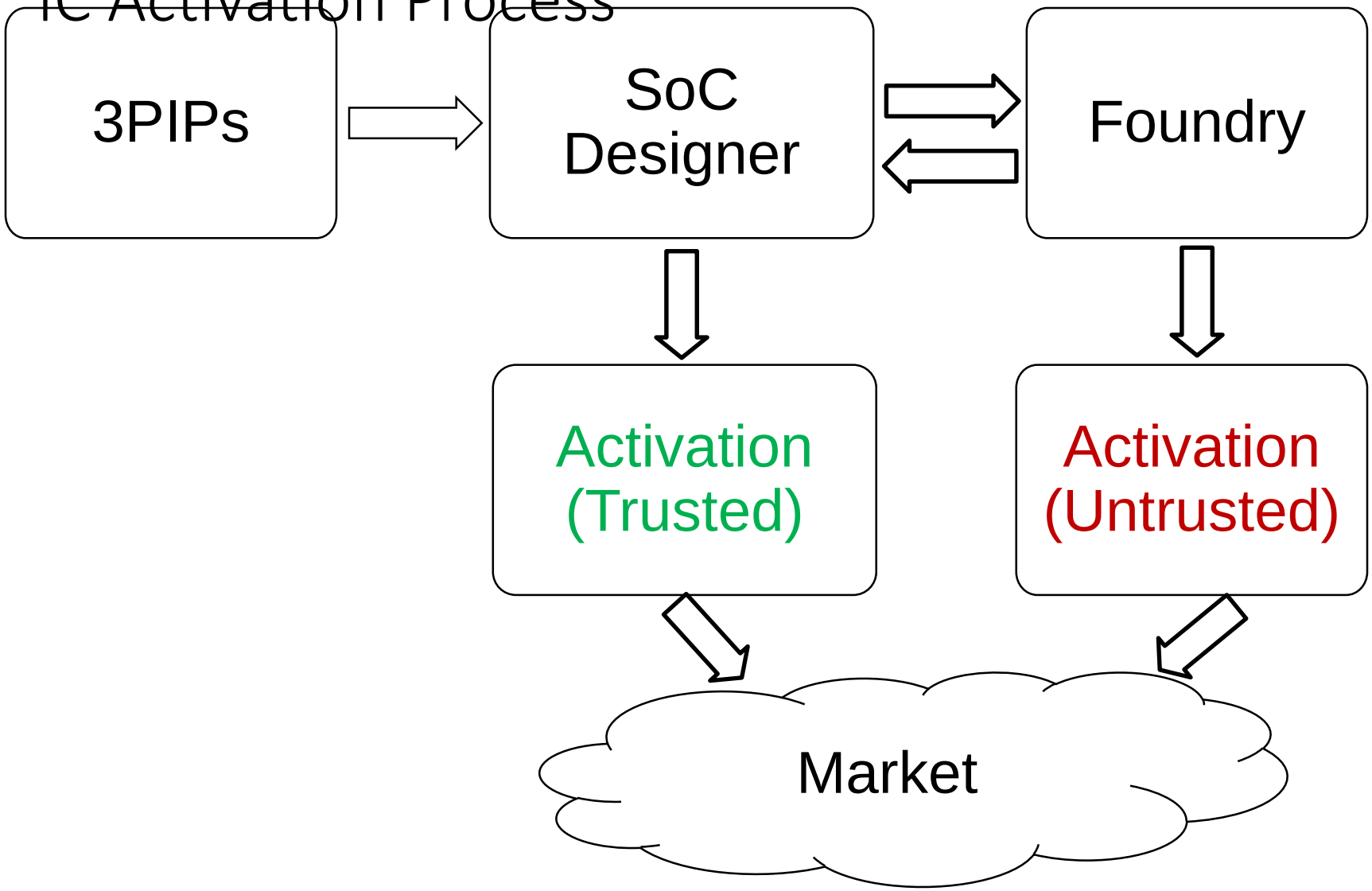
# Initial Proposal for Logic Locking



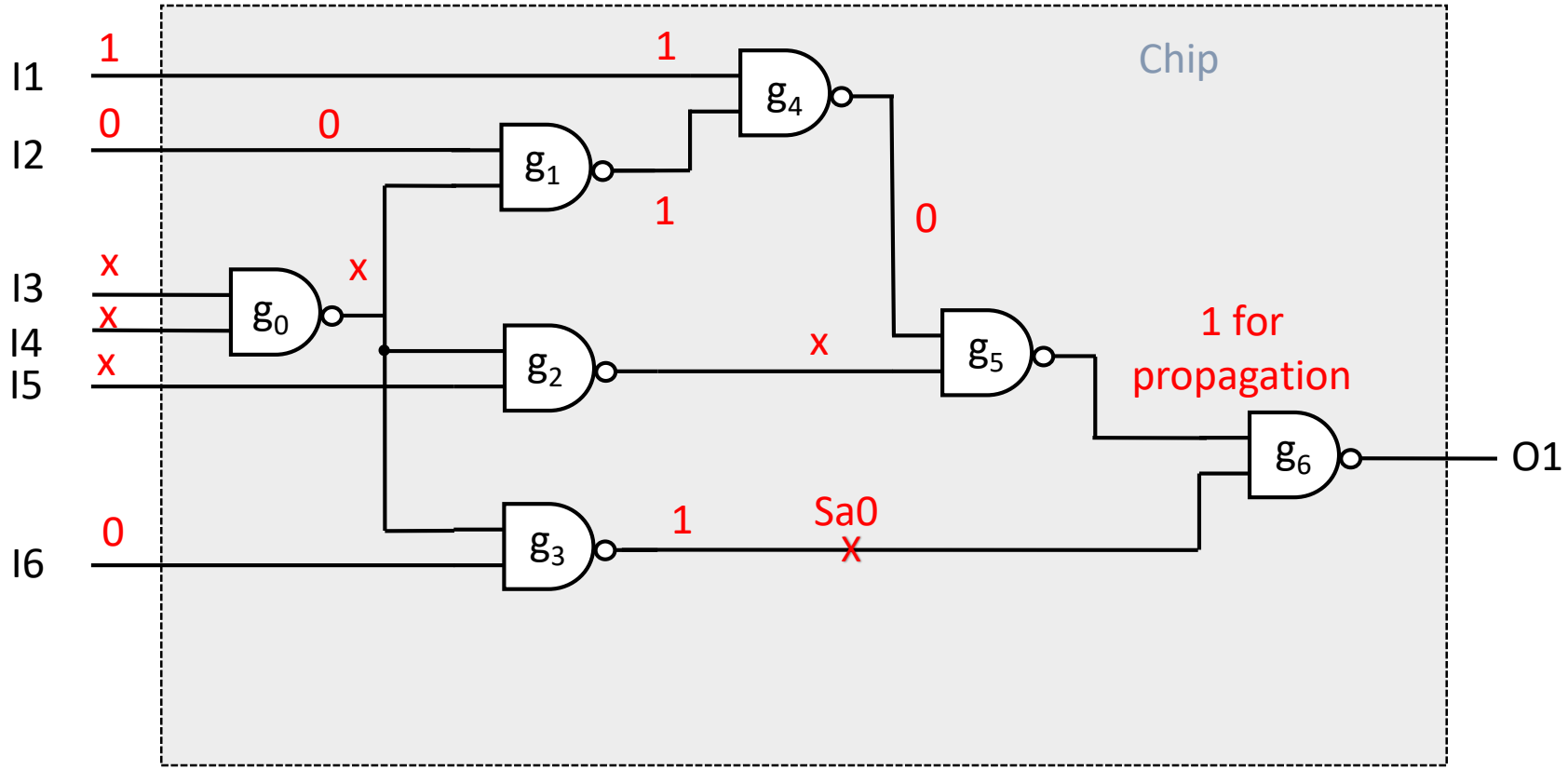
# Design-for-Security (DFS) Requirements

- No effect on functionality of the original circuit
- Attack resistance
  - Designs do not leak key during the manufacturing tests
  - Reverse engineering does not reveal key bits
- Structural test capability without the key
  - Manufacturing tests must be performed at the foundries and assemblies
- Post-silicon validation and debug capability
  - Full functional test capability at the secure site
- Full in-system test capability
- Reasonable overhead

# IC Activation Process

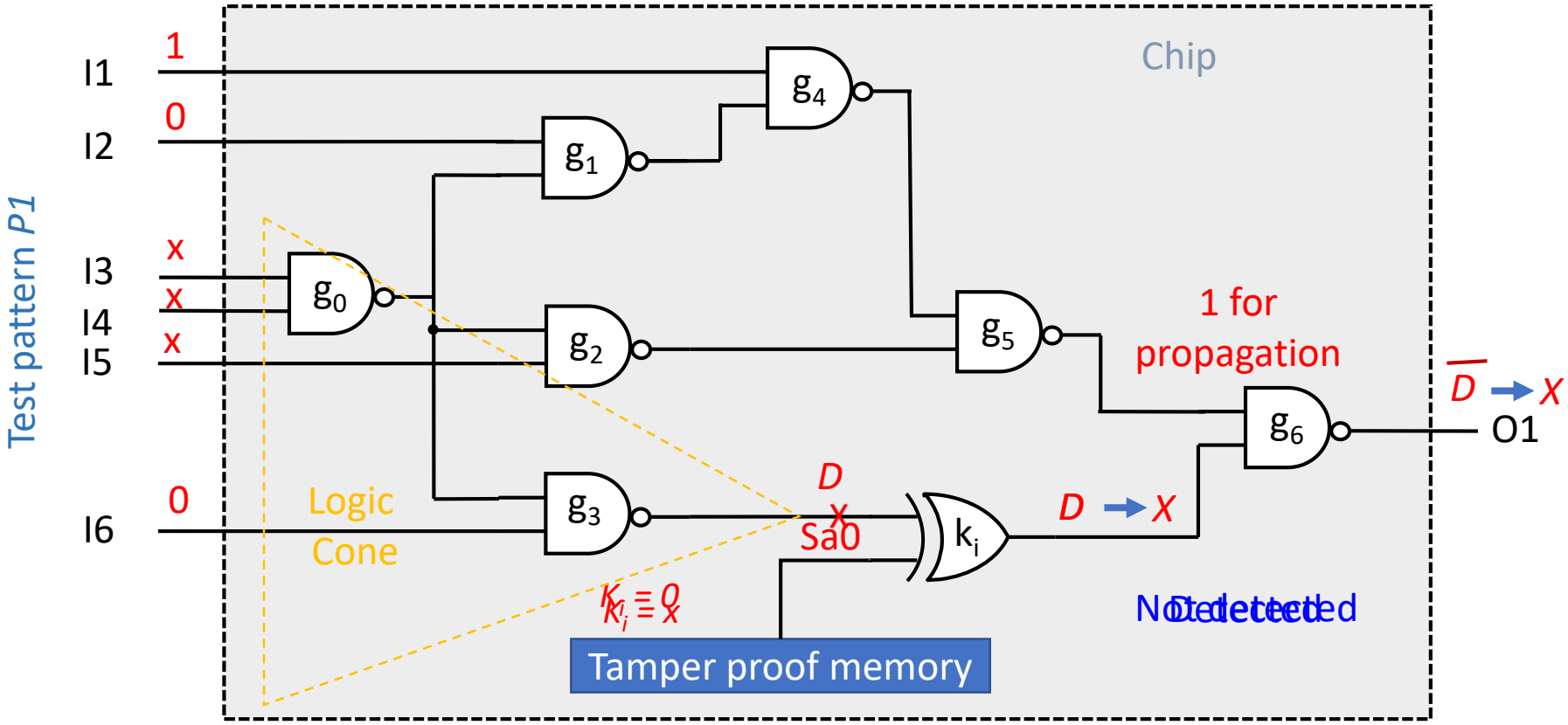


# Test Before Activation



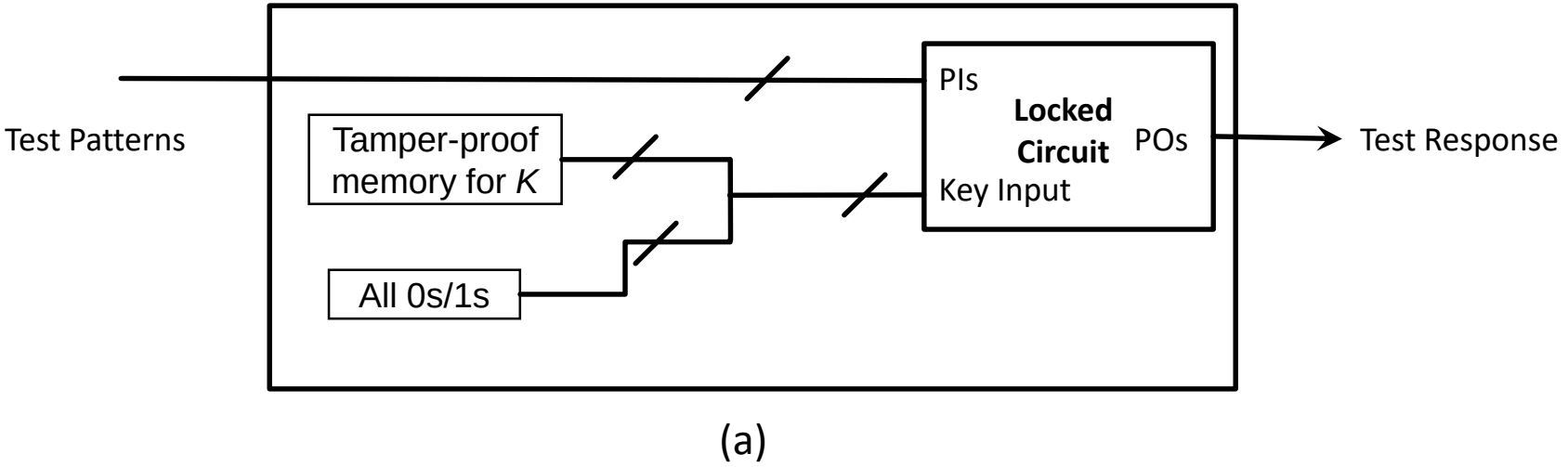

Generate test pattern from original circuit: *P1*

# Test Before Activation- Cont.

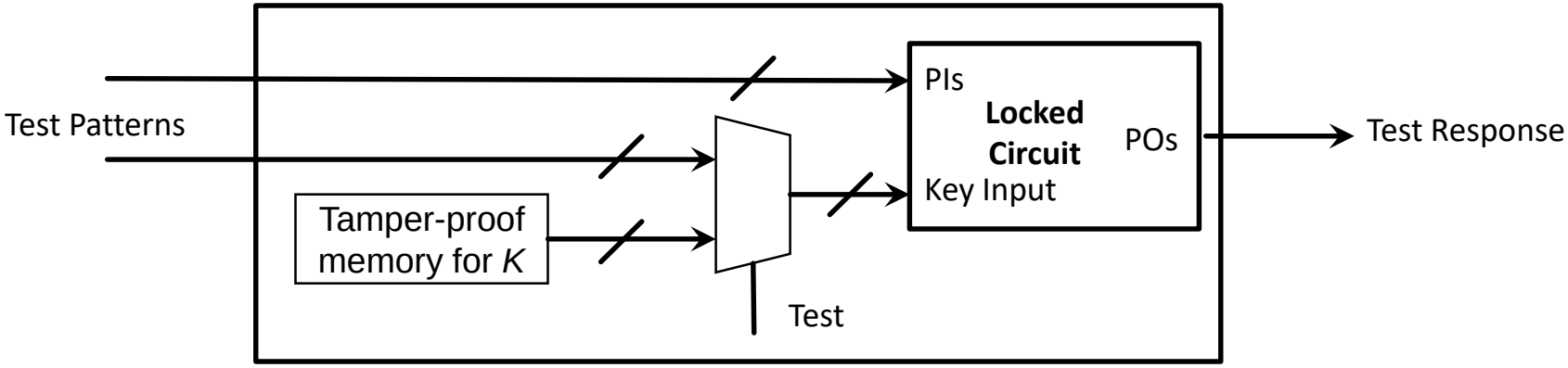



Need to know the key to generate test pattern

# Test Before Activation - Cont.




(a)




(b)

# Part II(a): Adversary Modeling

# Trump’s Wall: Design Intent

# Trump's Wall: Likely Reality?

Attacker

Asset

Enforcement Mechanism

# What is missing from this picture?

Attacker

Asset

Enforcement Mechanism

**Security property**: what is the goal of enforcement?
**Adversary model**: what can the attacker do while attempting to subvert the enforcement mechanism?

Attacker

Asset

Enforcement Mechanism

**Security property: what is the goal of enforcement?**

Adversary model: what can the attacker do while attempting to subvert the enforcement mechanism?

# Elaborating on the Security Property

Some example security properties:

1. No person is able to immigrate to the US without papers
2. No one is able to cross the Mexico-US border without papers
3. No undocumented immigrant is able to work without papers

These goals require very different enforcement mechanisms

# An Aside: Fuzzy vs. Well-defined Security

**Fuzzy security.** By *fuzzy security* I mean the following process: some guy comes up with some sort of cryptographic algorithm. He then makes some vague claims about the security of this algorithm, and people start using it for applications of national or personal security of the utmost importance. Then, someone else (a hacker) manages to break this algorithm, usually with disastrous results to its users, and then the inventor or users either "tweak" the algorithm, hoping that the new tweak is secure, or one invent a new algorithm. The distinguishing mark of fuzzy security is *not* that it is often broken with disastrous outcomes. This is a side effect. The distinguishing mark is that there is **never a rigorous definition of security, and so there is never a clearly stated conjecture of the security properties** of this algorithm.

Boaz Barak

https://www.boazbarak.org/papers/obf_informal

Attacker

Asset

Enforcement Mechanism

Security property: what is the goal of enforcement?

**Adversary model: what can the attacker do while attempting to subvert the enforcement mechanism?**

# Elaborating on the attacker model

- Can the attacker use ladders?
- Can the attacker fly?
- How about tunnel?
- Can attacker cross legally but overstay their visa?

**My goal is not to dunk on the wall, but to point out that we can't evaluate whether a scheme is secure without a clear definition of adversary capabilities**

# What is an Adversary Model?

Precise statement of adversary capabilities that is used to systematically reason about the security of a particular protocol or enforcement mechanism

# Logic Locking: Security Property

Only authorized users can operate locked IC

Implicit in definition

- Some notion of unauthorized user
- What operate means

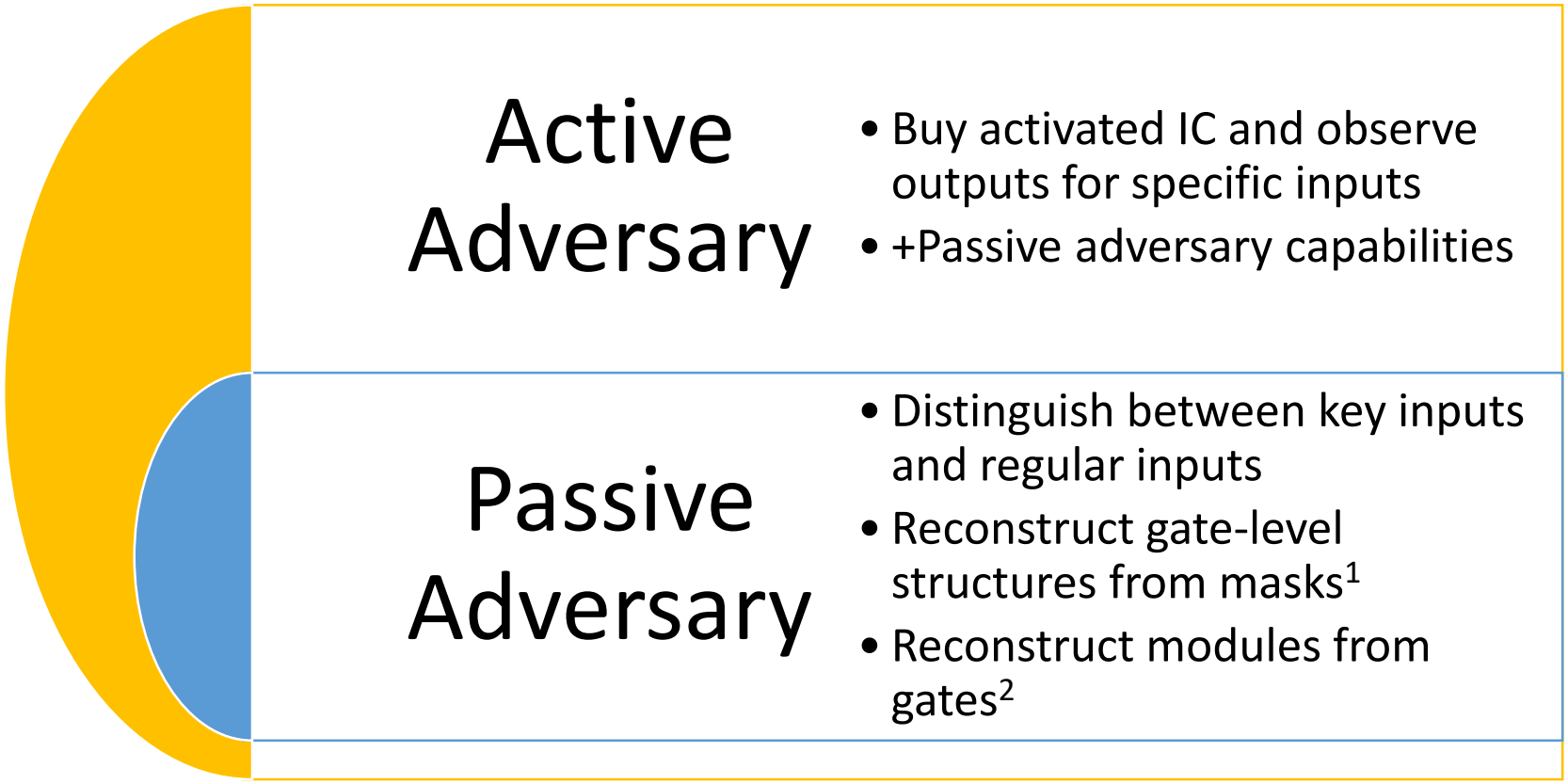
# Logic Locking: Adversary Model

Want to capture abilities of malicious foundry

What can foundry do?

- Distinguish between key inputs and regular inputs? 
- Reverse engineer gate-level netlist from masks? 
- Further analysis to obtain-high level structures? 
- Buy an activated IC from the market and observe outputs for specified inputs? (I/O Oracle access) **Maybe**

# We'll Consider Two Adversaries



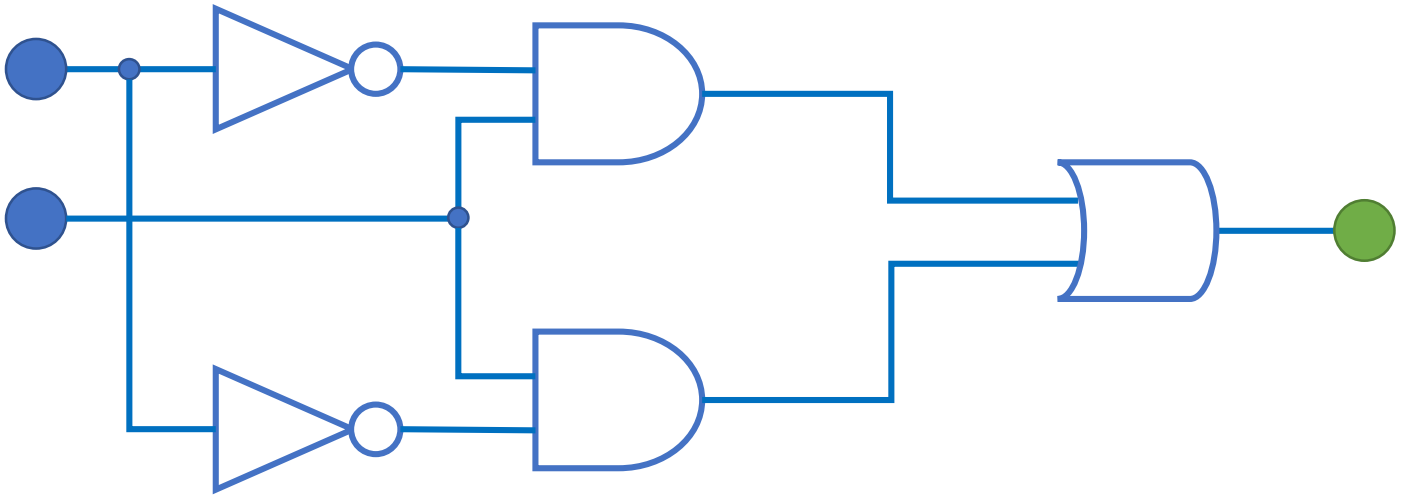

[1][Torrance et al., CHES'09], [2][Subramanyan et al., TETC'14]

# Can you do anything without Oracle access?

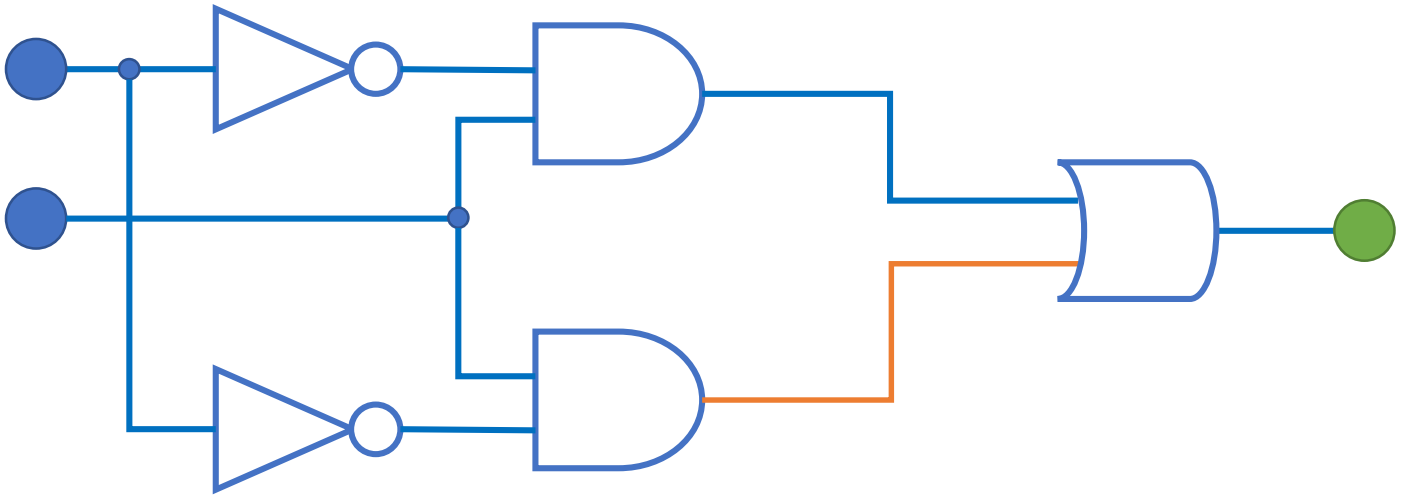
**Strawman locking algorithm**

1. Pick a random wire in the circuit
2. Choose a random gate $\in \{\mathrm{AND2}, \mathrm{OR2}\}$
3. Connect one input of gate to the wire's driver
4. Connect the other input of gate to a key input
5. Connect the output of the gate to the wire's load
6. Correct key input = 1 if inserted gate was AND2 and 0 otherwise
7. Repeat 1-6 as many times as desired

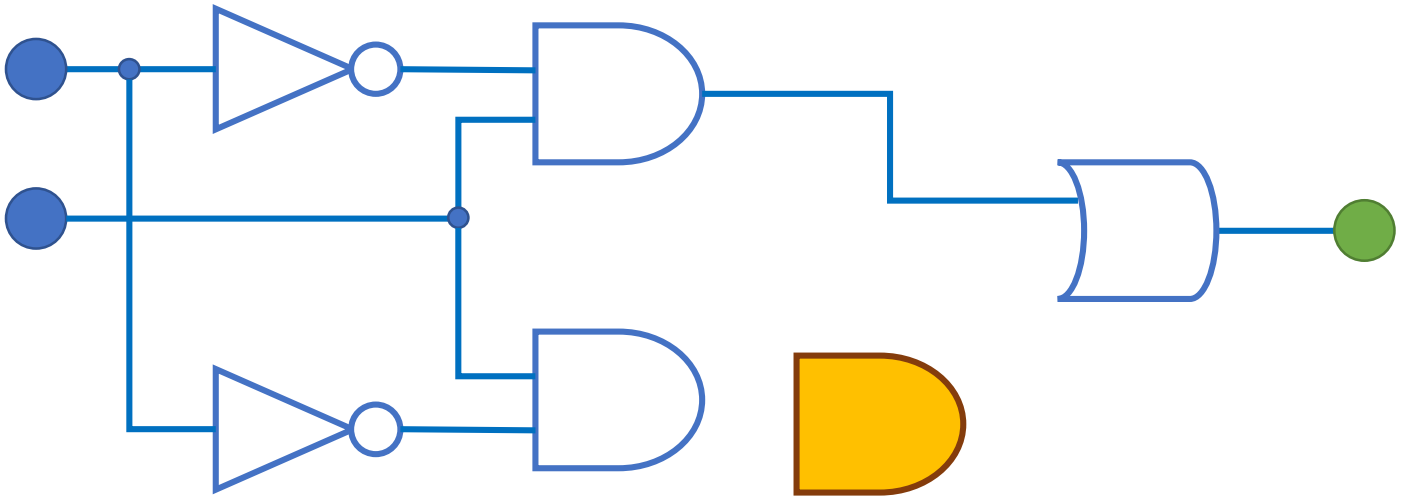
# Security Analysis of Strawman Locking Algo



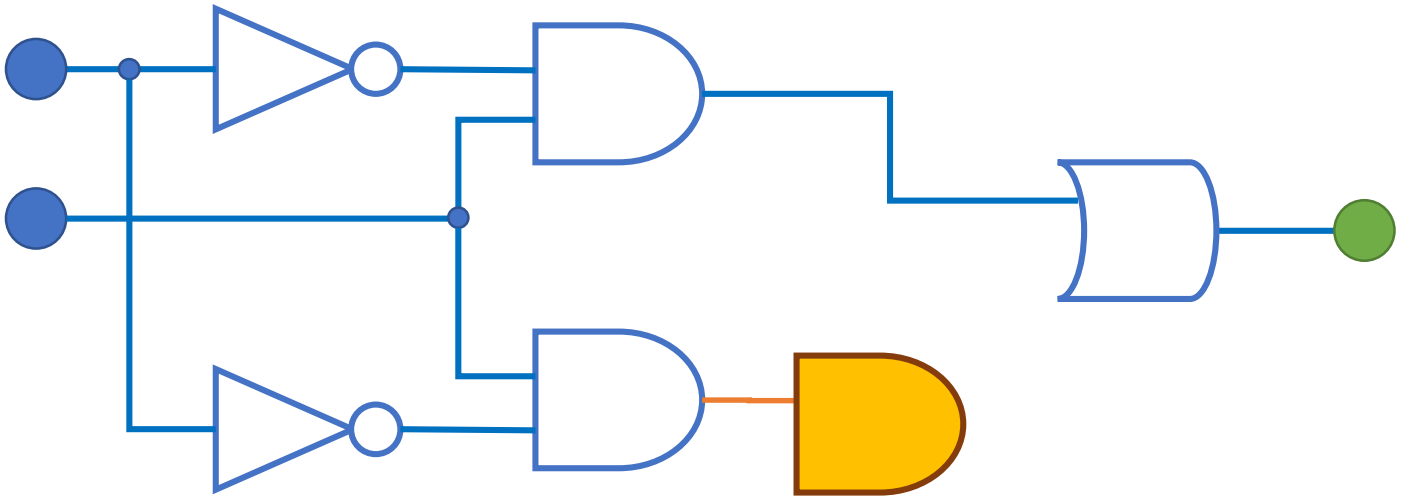
# Security Analysis of Strawman Locking Algo



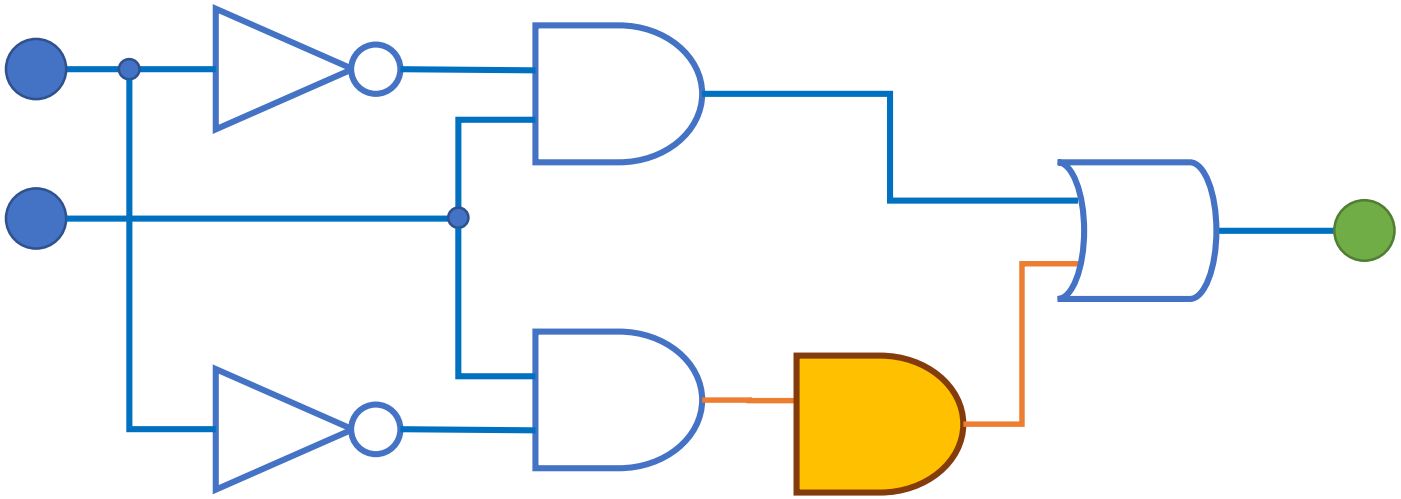
# Security Analysis of Strawman Locking Algo



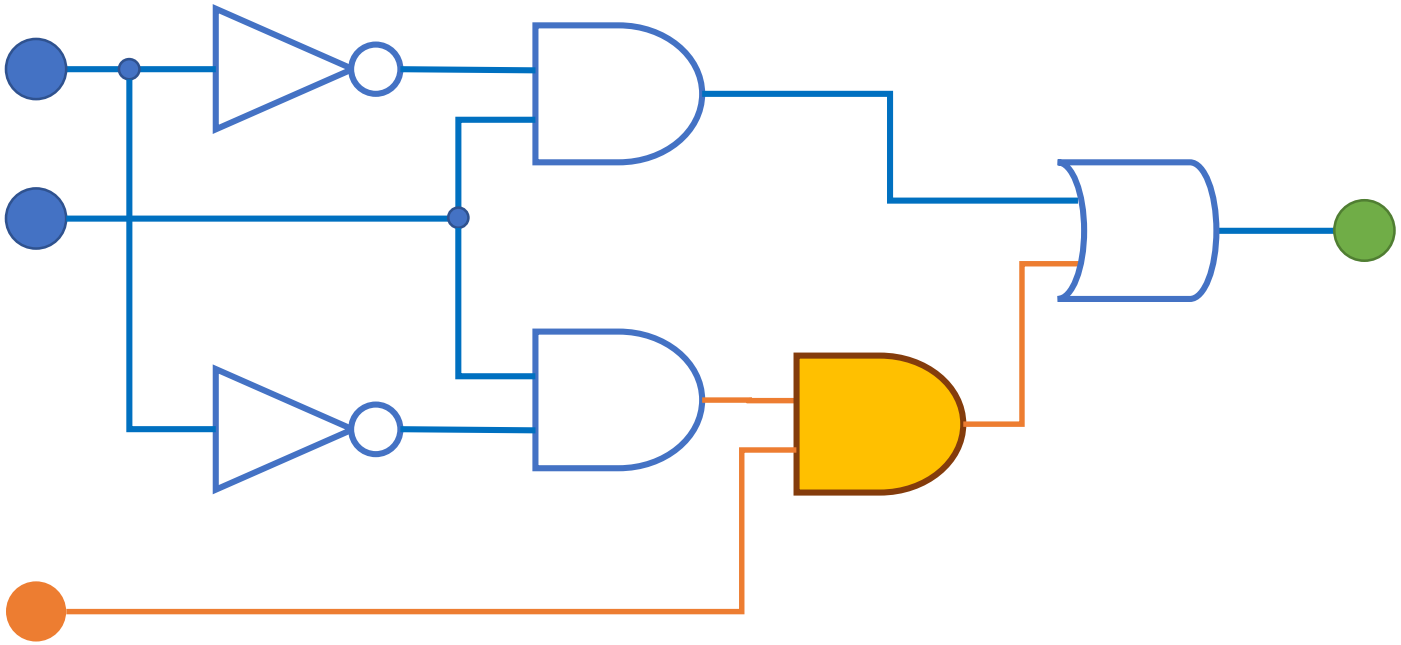
# Security Analysis of Strawman Locking Algo



# Security Analysis of Strawman Locking Algo

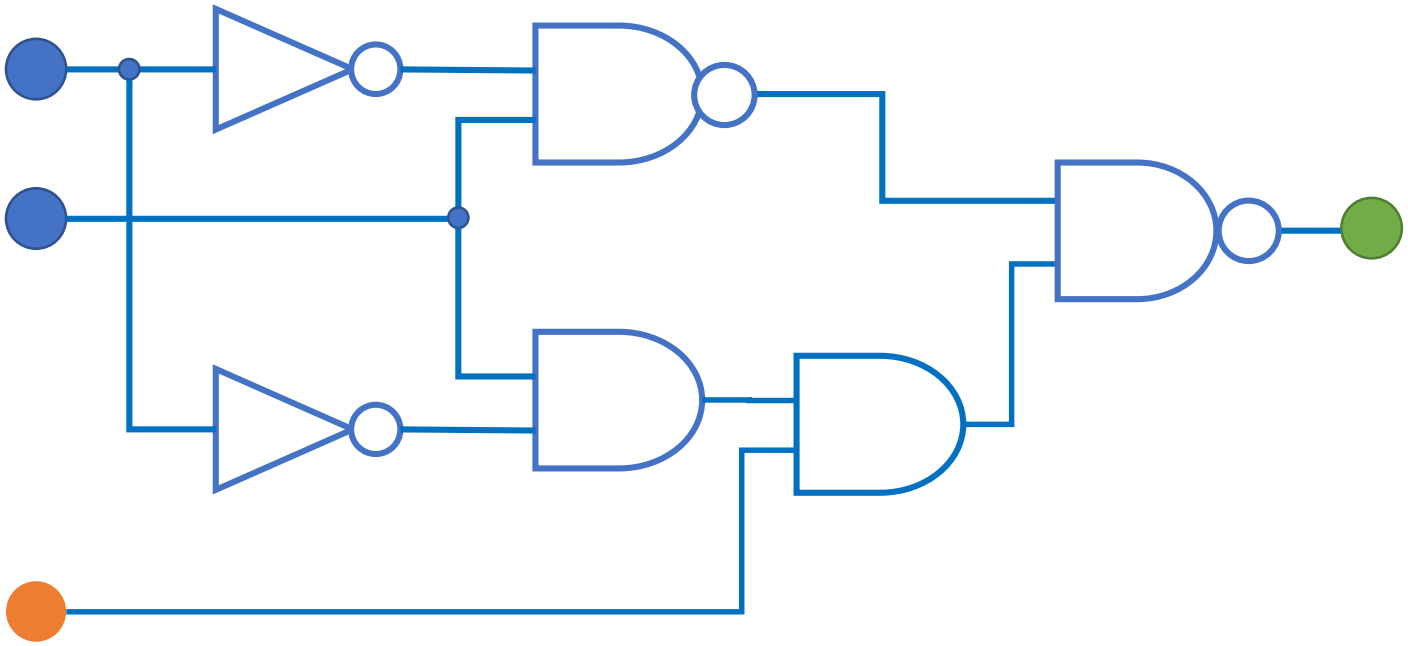


# Security Analysis of Strawman Locking Algo



Key Input

# What is the key input value?



Key Input

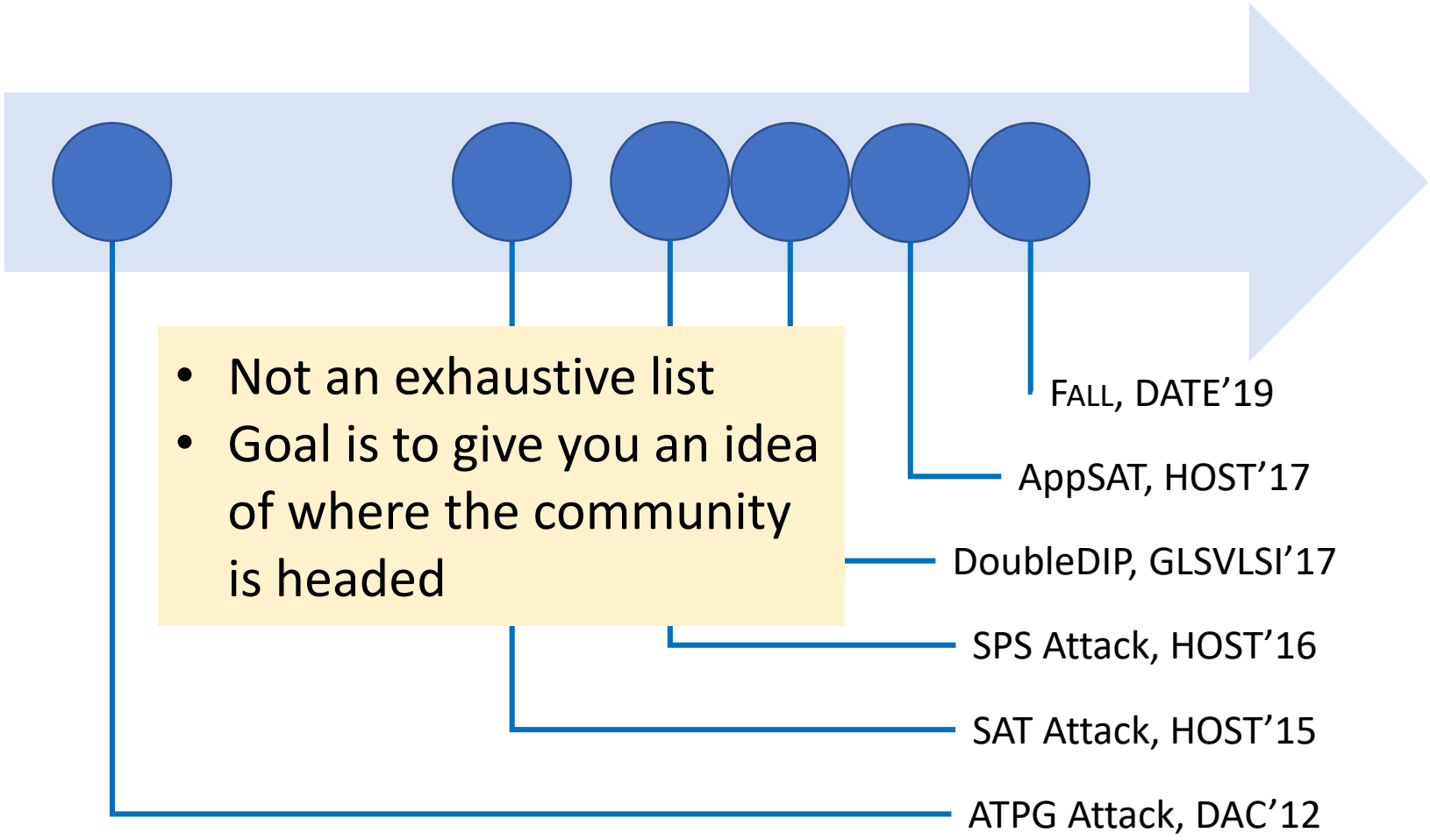
AND gates connected to key inputs means key=1!

# Morals of the Story

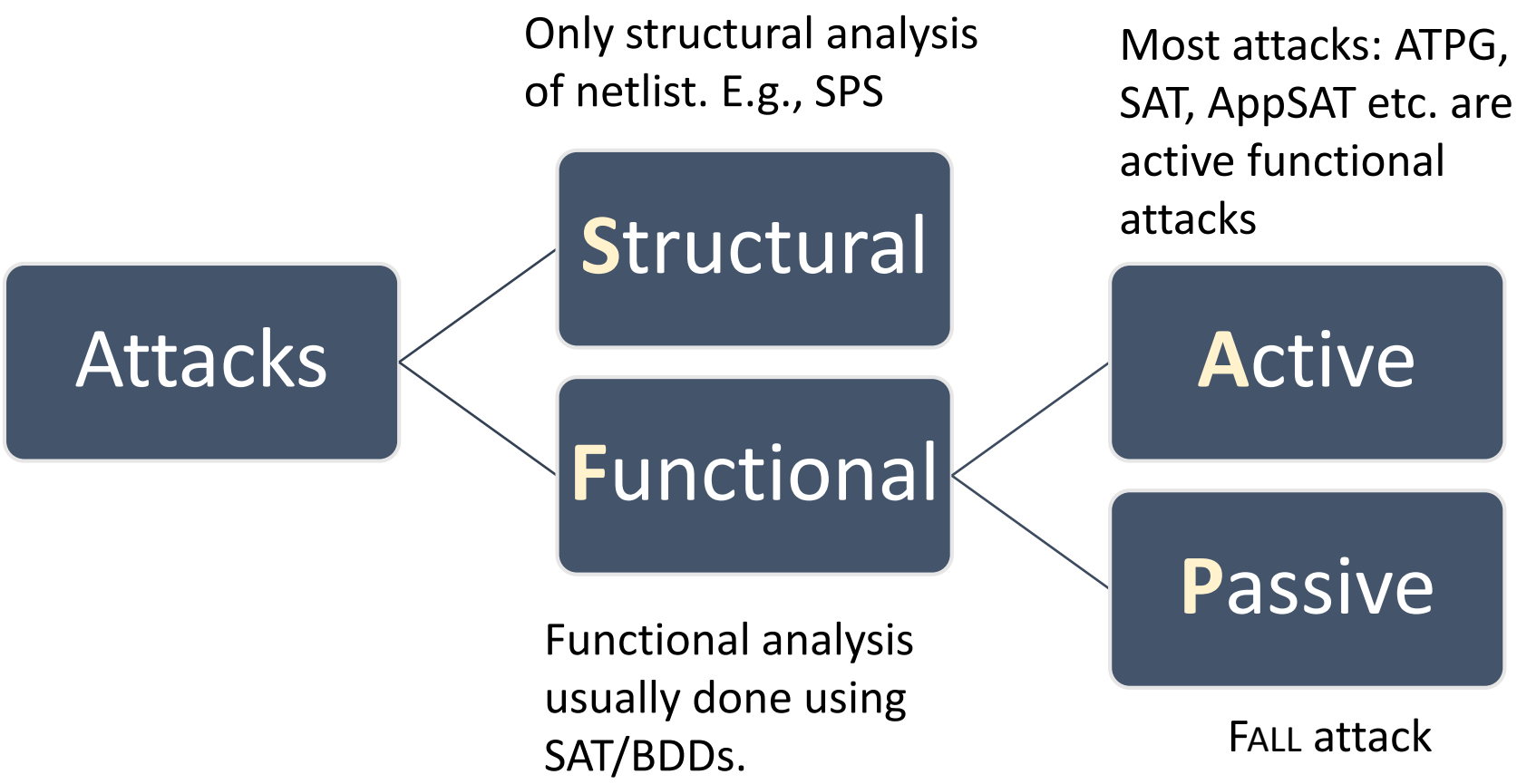
- Careful definition of security property and adversary model are very important
- Even seemingly weak adversaries can break insecure locking algorithms

# Part II(b): Attacks on Logic Locking

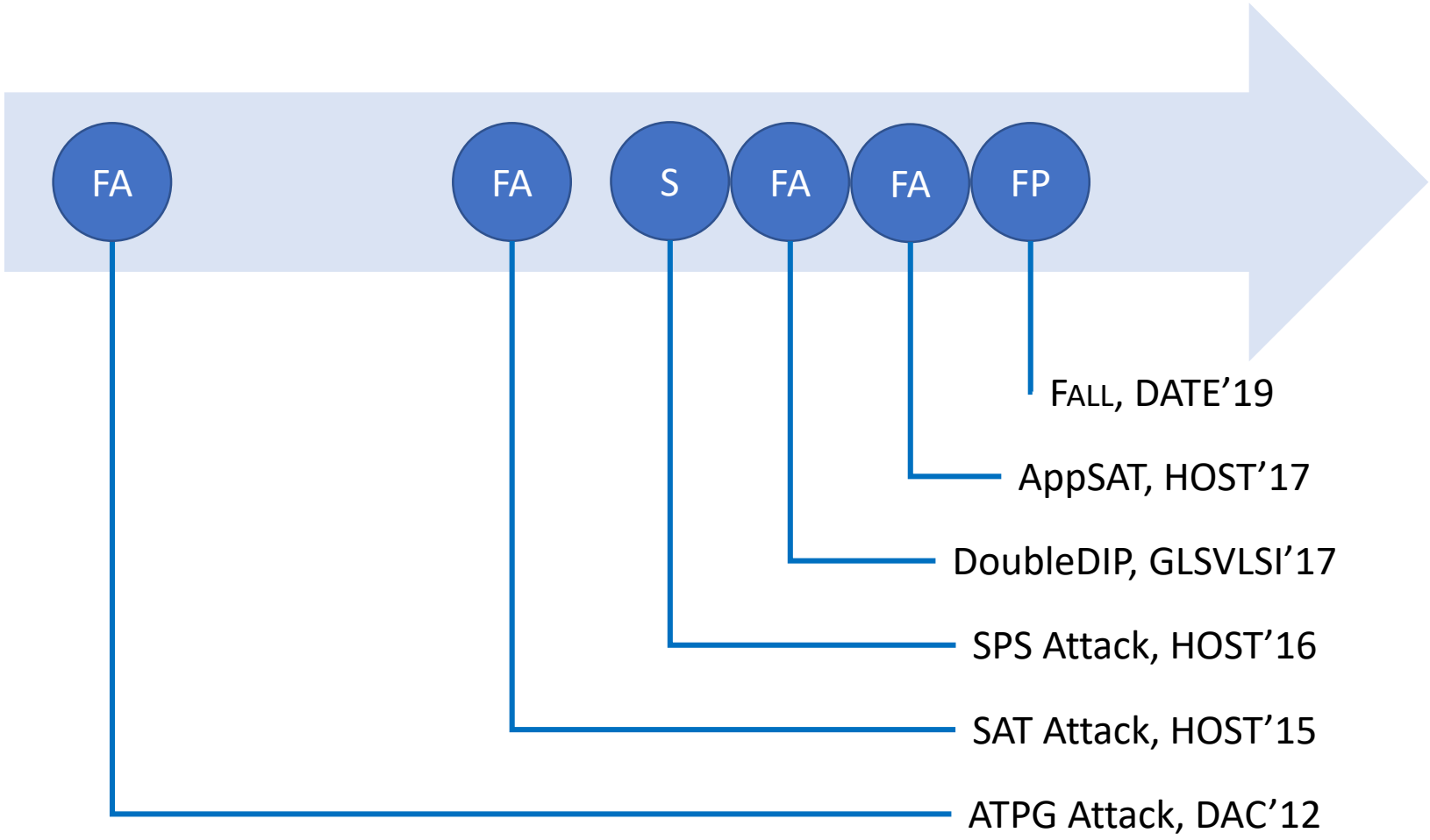
# A Timeline of Attacks



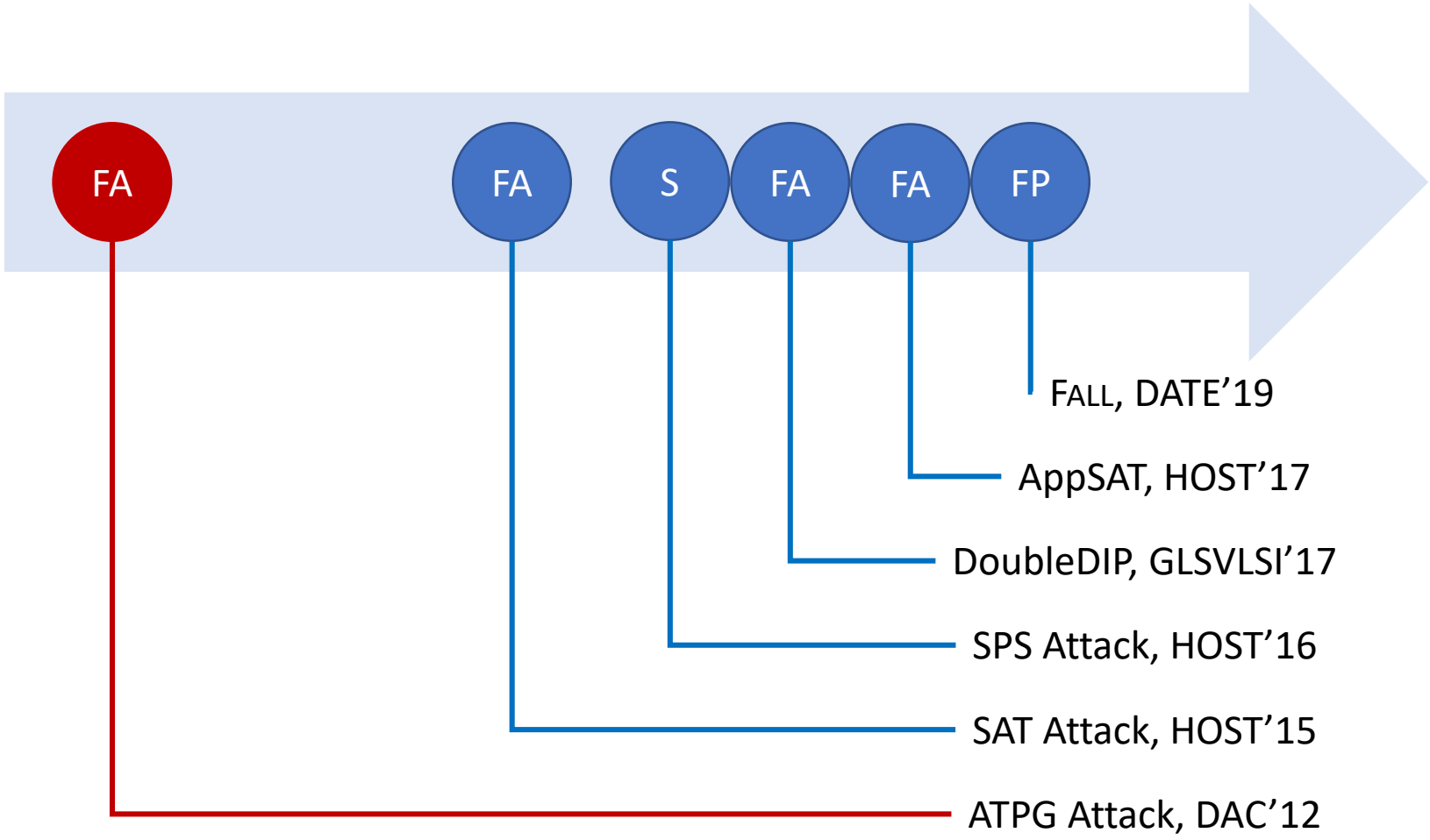
# Classification of Attacks



# A Timeline of Attacks

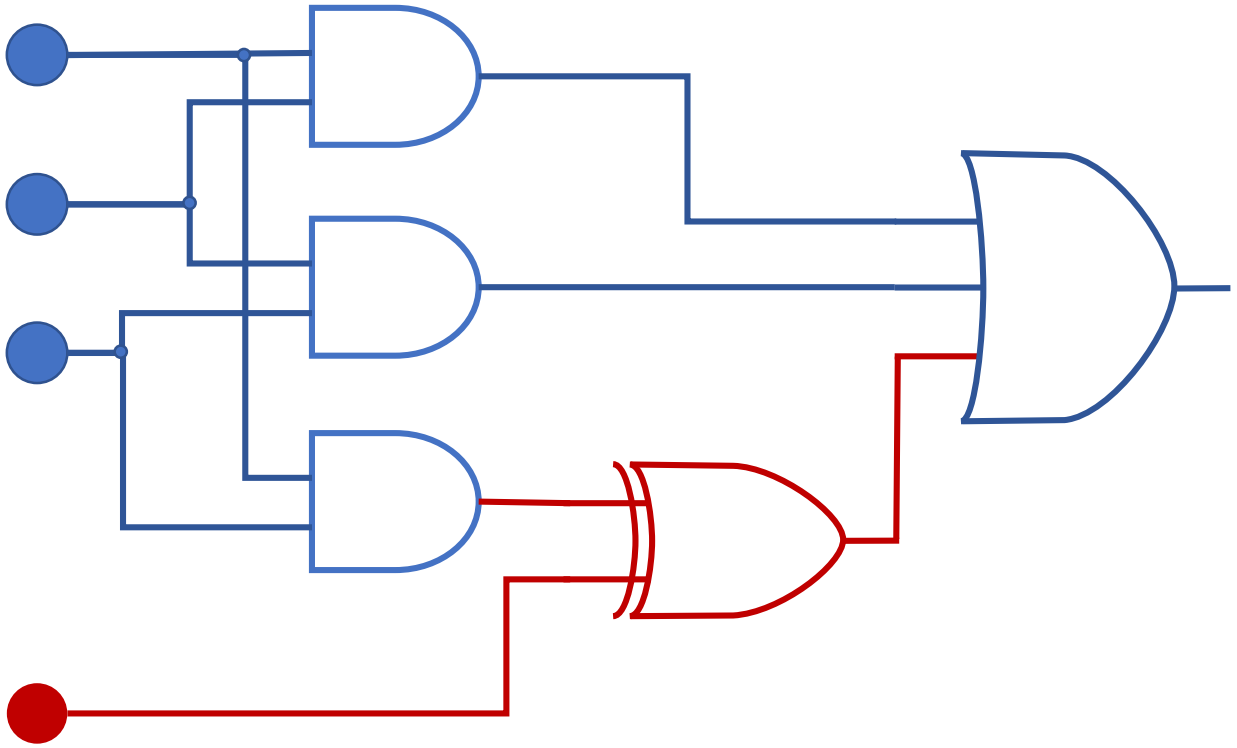


# ATPG Attack


FA
FA
S
FA
FA
FP
FALL, DATE'19
AppSAT, HOST'17
DoubleDIP, GLSVLSI'17
SPS Attack, HOST'16
SAT Attack, HOST'15
ATPG Attack, DAC'12

# ATPG Attack

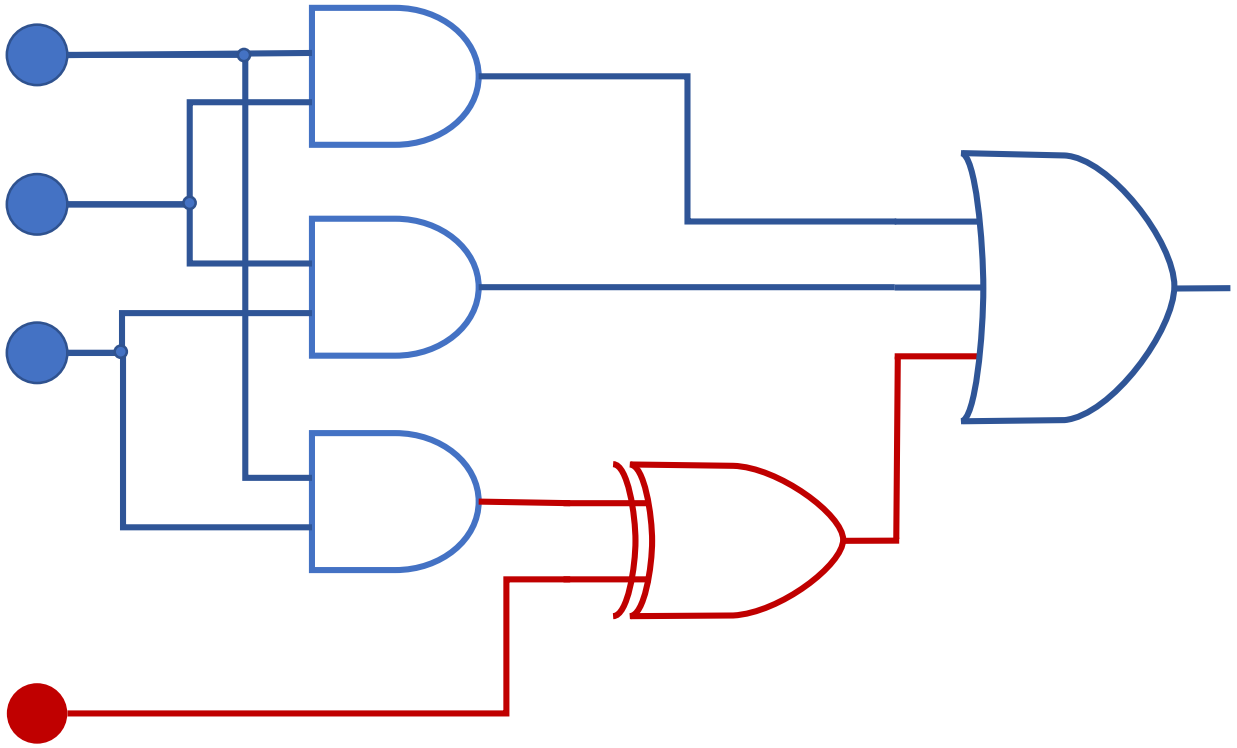
[Rajendran et al., DAC'12]



Q: What is the value of the key input?

# ATPG Attack

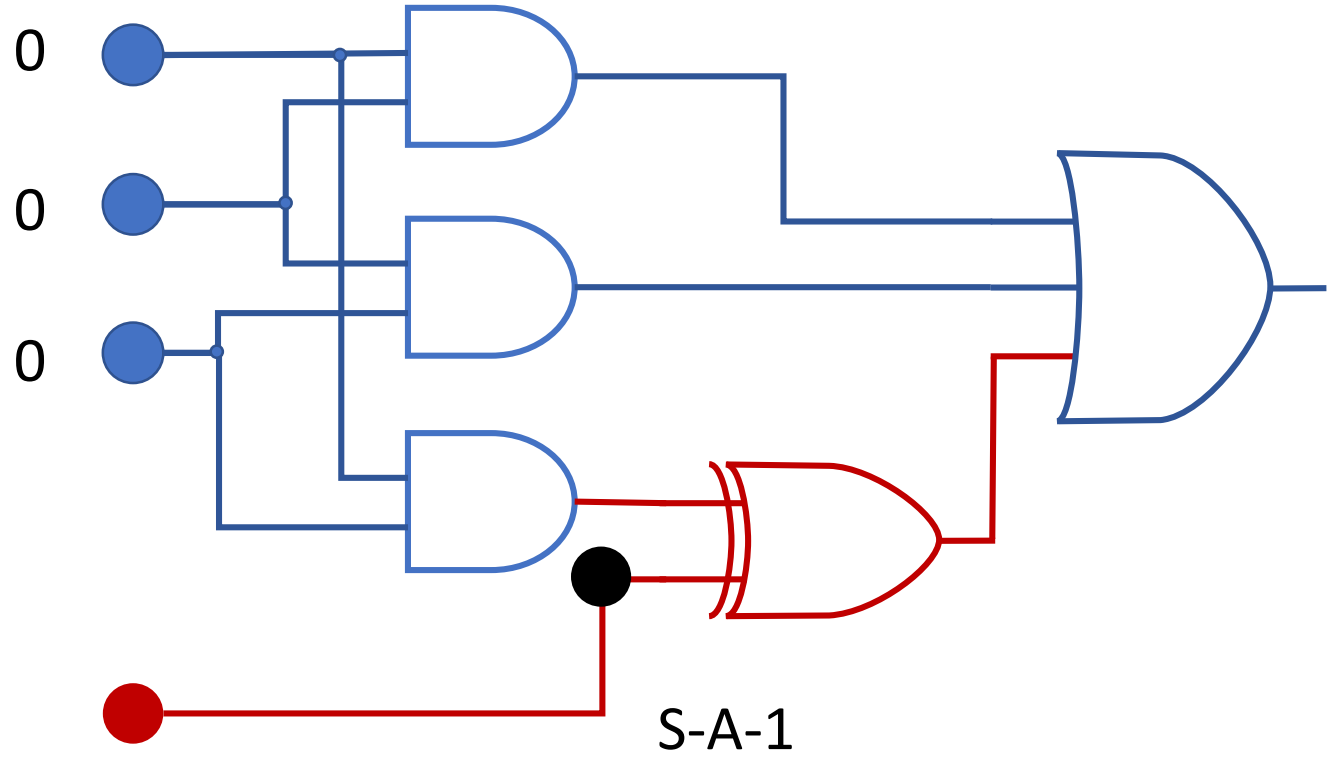
[Rajendran et al., DAC'12]



Insight: view key input like a stuck-at fault

# ATPG Attack

[Rajendran et al., DAC'12]




Input vector which exposes fault will reveal key

# ATPG Attack Algorithm

[Rajendran et al., DAC'12]

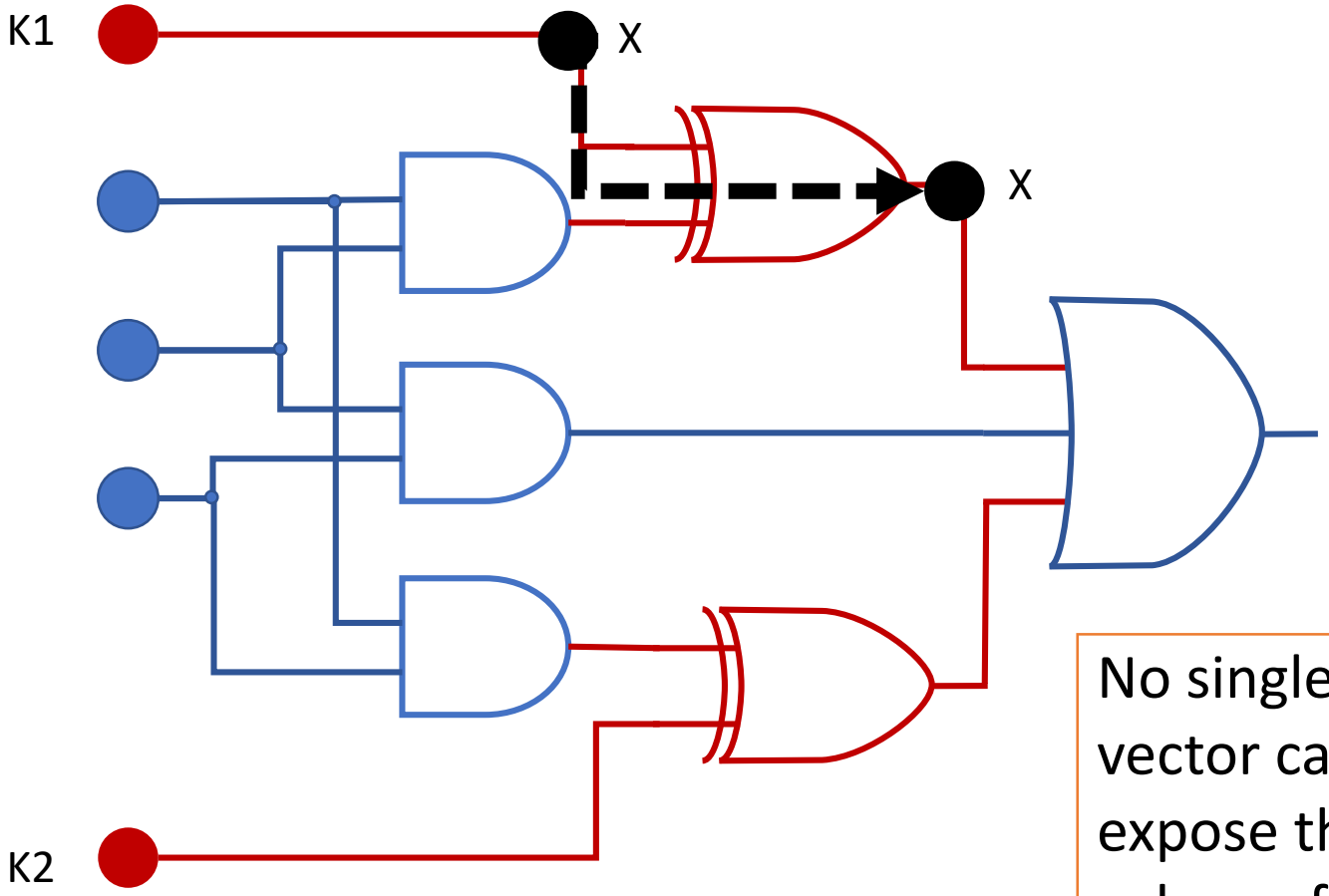
Input: netlist, key input to attack

Procedure:

- Set unknown key inputs to X, known key inputs to appropriate values
- Find test vector to expose stuck-at fault at output of gate connected to this key input
- Apply this test vector on the activated IC
- Output will determine key value

Repeat for remaining key inputs

# ATPG Attack Countermeasure [Rajendran et al., DAC'12]




No single input vector can expose the values of K1/K2

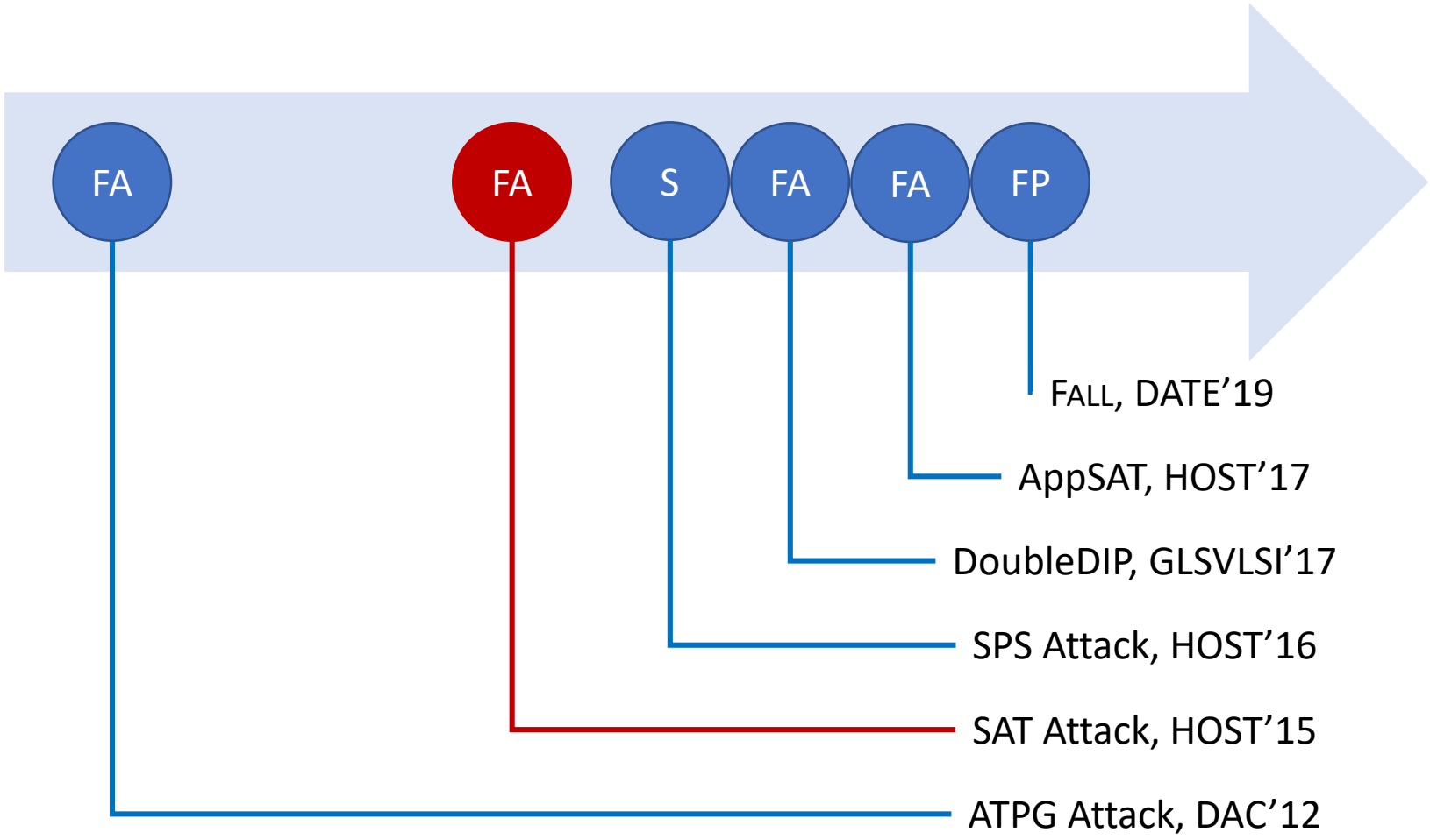
# ATPG Attack Summary

[Rajendran et al., DAC'12]

- Key idea: find vectors that propagate key inputs to the output
- ATPG tools solve exactly the above problem, so utilize them
- Countermeasure: prevent propagation by making key inputs interfere

# The SAT Attack

[Subramanyan et al., HOST'15]



# A Brief Interlude

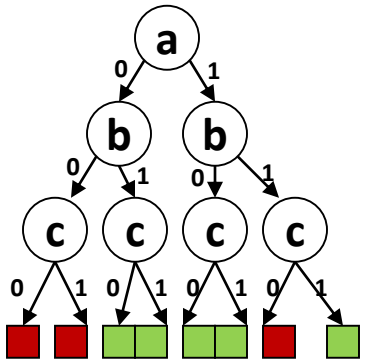
## SAT Solving

# What is SAT Solving?

Given a propositional logic (Boolean) formula, find a variable assignment such that the formula evaluates to 1, or prove no such assignment exists.

$$\mathbf{F = (a \vee b) \ \wedge \ (\neg a \vee \neg b \vee c)}$$

For *n* variables, there are $2^n$ possible truth assignments to be checked.



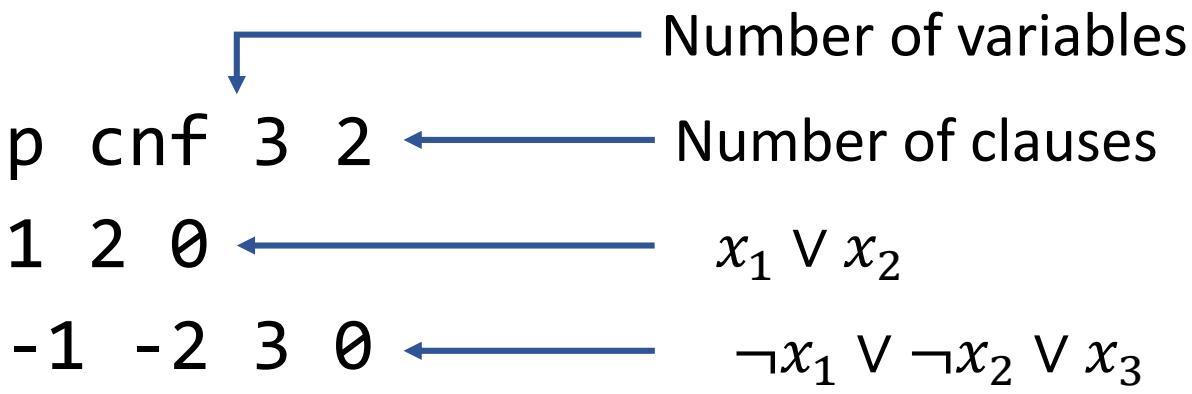

First established NP-Complete problem.

S. A. Cook, The complexity of theorem proving procedures, *Third Annual ACM Symposium on the Theory of Computing*, 1971

# Aren't NP-Complete Problems Hard?

- Modern solvers can **regularly** solve **practical SAT** instances with **millions of variables and constraints**
- Practical impact of SAT solvers
  - Electronic Design Automation (EDA): logic synthesis, equivalence checking, assertion checking, post-silicon validation
  - Software verification: core of most program verification techniques; Regularly used at MS, Google, Amazon, FB etc.
  - AI/Planning: Used in many constraint solving procedures

# SAT Solver Usage

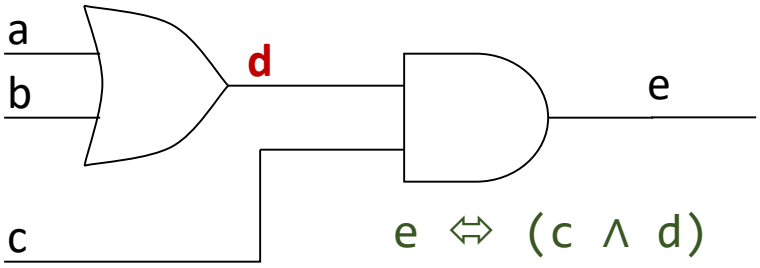



- Input is a formula in CNF
- Output is an assignment to the variables or UNSAT
- Example shown above is in DIMACS format

# Using SAT Solvers

- Accept input in CNF (product of sums, conjunction of disjunctions)
- Obviously, circuits in general are not in CNF
- The naïve conversion to CNF could result in an exponential blowup
- But there is a way out, every circuit can be efficiently encoded in CNF
- Trick is to introduce (polynomially many) new variables

# Converting to CNF: Tseitin Transformation




$d \Leftrightarrow (a \vee b)$

$(a \vee b \vee \neg d) \wedge$
$(\neg a \vee d) \wedge$
$(\neg b \vee d) \wedge$

$e \Leftrightarrow (c \wedge d)$

$(\neg c \vee \neg d \vee e) \wedge$
$(d \vee \neg e) \wedge$
$(c \vee \neg e) \wedge$

**Tseitin Transformation: Procedure**

- Create a new variable for each gate's output in the circuit
- Create clauses that capture functionality of each gate (see next slide for anohter example)
- Conjunction of all clauses is the desired CNF formula

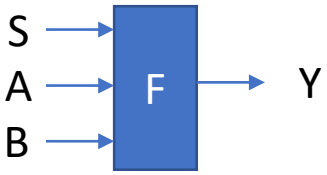
Can 'e' ever become true?

Is $(e) \wedge (a \vee b \vee \neg d) \wedge (\neg a \vee d) \wedge (\neg b \vee d) \wedge (\neg c \vee \neg d \vee e) \wedge (d \vee \neg e) \wedge (c \vee \neg e)$ satisfiable?

Circuit C with inputs X, outputs Y, internal gates Z is represented as C(X, Y, Z) or C(X, Y)

# One More Example

| S | A | B | Y |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 |




What is this function?

- What is Y when S = 0?
- What about when S = 1?

$$(\neg s \wedge \neg a \wedge \neg b \rightarrow \neg y) \wedge$$
$$(\neg s \wedge \neg a \wedge b \rightarrow \neg y) \wedge$$
$$(\neg s \wedge a \wedge \neg b \rightarrow y) \wedge$$
$$(\neg s \wedge a \wedge b \rightarrow y) \wedge$$
$$(s \wedge \neg a \wedge \neg b \rightarrow \neg y) \wedge$$
$$(s \wedge \neg a \wedge b \rightarrow \neg y) \wedge$$
$$(s \wedge a \wedge \neg b \rightarrow y) \wedge$$
$$(s \wedge a \wedge b \rightarrow y)$$

$$(F \wedge \neg x \rightarrow p) \wedge$$
$$(F \wedge x \rightarrow p) \wedge$$
$$\equiv (F \rightarrow p)$$

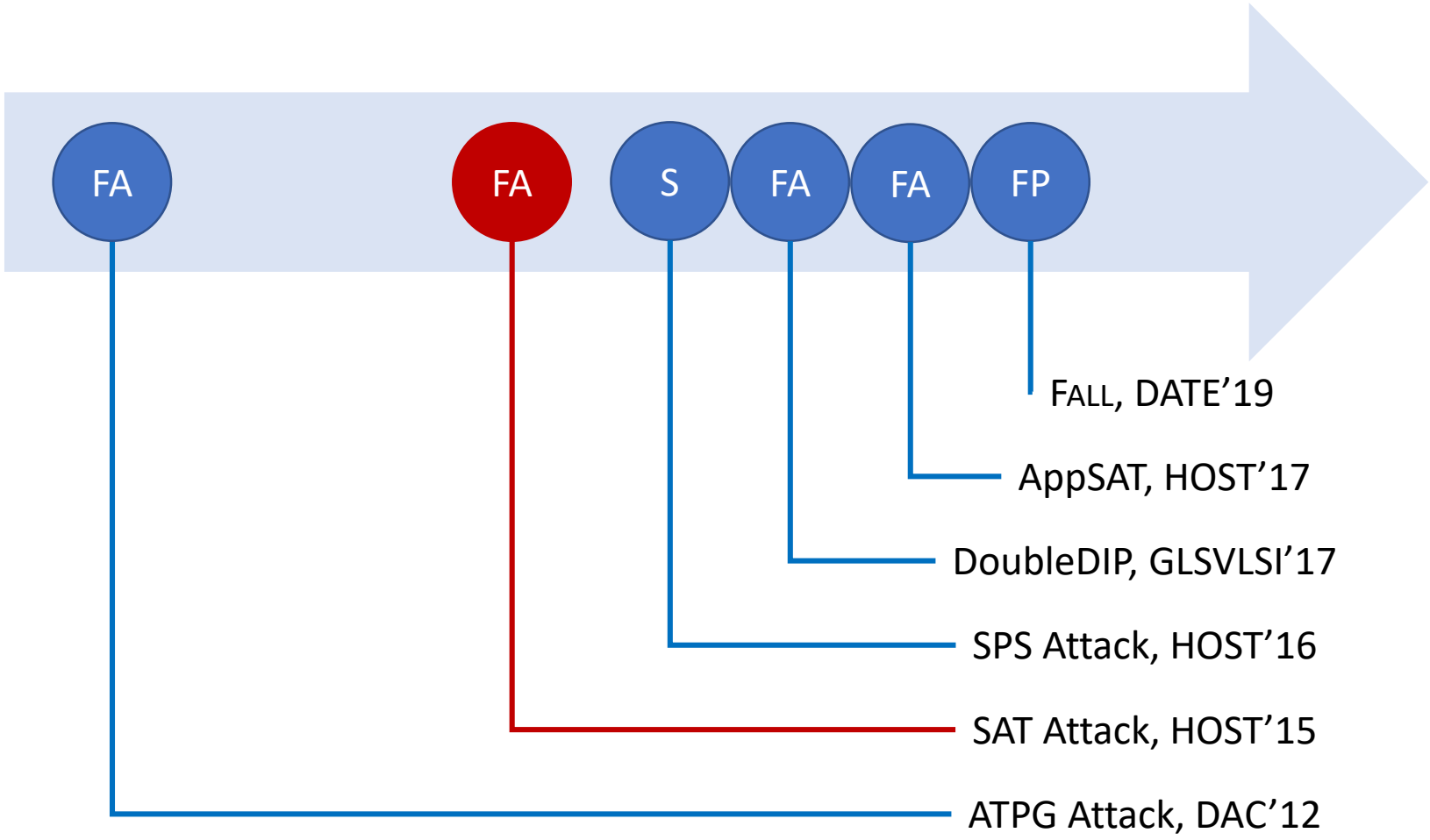
$$(A \rightarrow B) \equiv (\neg A \vee B)$$

$$(\neg s \wedge \neg a \rightarrow \neg y) \wedge$$
$$(\neg s \wedge a \rightarrow y) \wedge$$
$$(s \wedge \neg b \rightarrow \neg y) \wedge$$
$$(s \wedge b \rightarrow y)$$

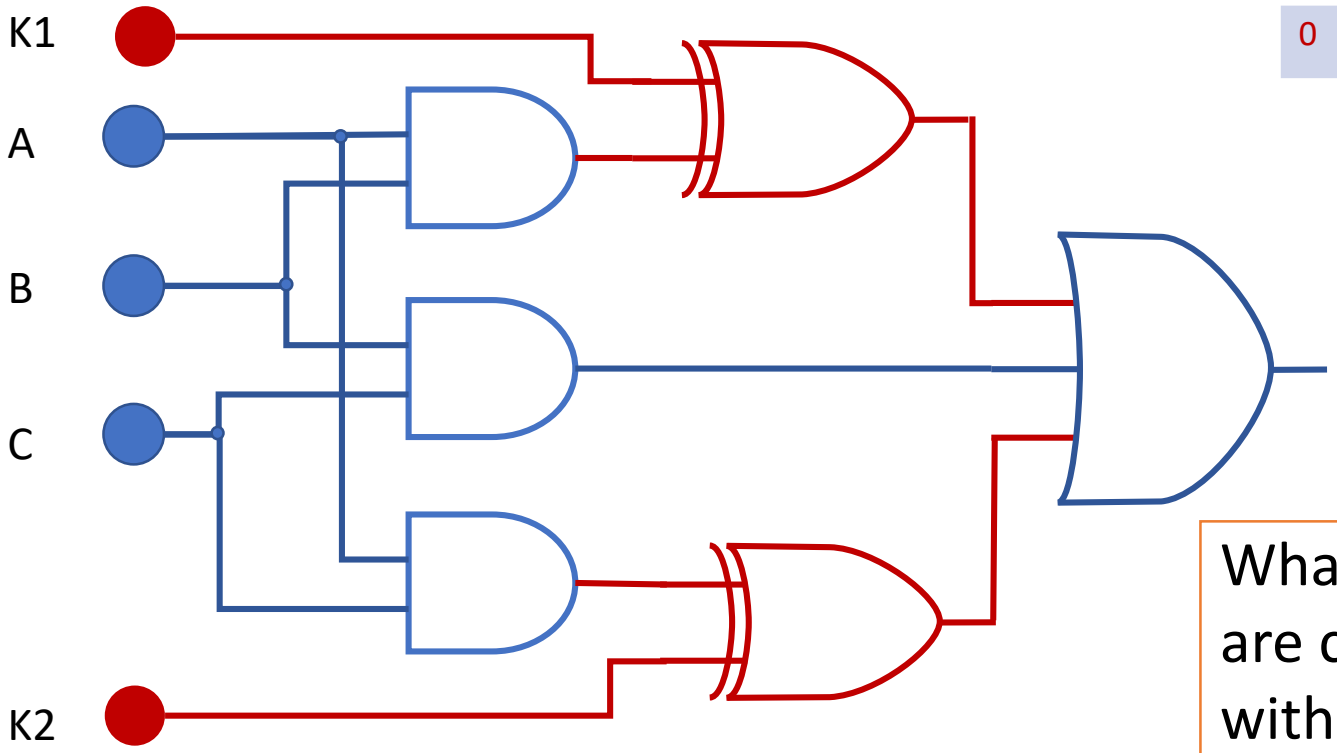
$$(s \vee a \vee \neg y) \wedge$$
$$(s \vee \neg a \vee y) \wedge$$
$$(\neg s \vee b \vee \neg y) \wedge$$
$$(\neg s \vee \neg b \vee y)$$

# The SAT Attack

[Subramanyan et al., HOST'15]



# Circuit is not actually secure!




| A | B | C | Y |
|---|---|---|---|
| 0 | 0 | 0 | 0 |

What key values are consistent with the above I/O patter?

Only (K1,K2)=(0,0)

# Main Idea in the SAT Attack

- Do the reasoning shown on the previous slide using the SAT solver
- Challenges
  - How do we generate the input vectors?
  - When do we know that we have the correct key?

# Strawman SAT attack

Suppose we have a locked circuit represented as C(X, K, Y)

- X: circuit inputs, K: key inputs, Y: outputs

**Strawman attack algorithm**

- Generate a lot of random inputs: $X_1, X_2, \ldots, X_N$
- Evaluate the output for each on activated IC: $Y_1, Y_2, \ldots, Y_N$
- Construct SAT formula: $C(X_1, K, Y_1) \wedge \cdots \wedge C(X_2, K, Y_2)$
- Satisfying assignment to $K$ is the key

# Does Strawman SAT attack work?

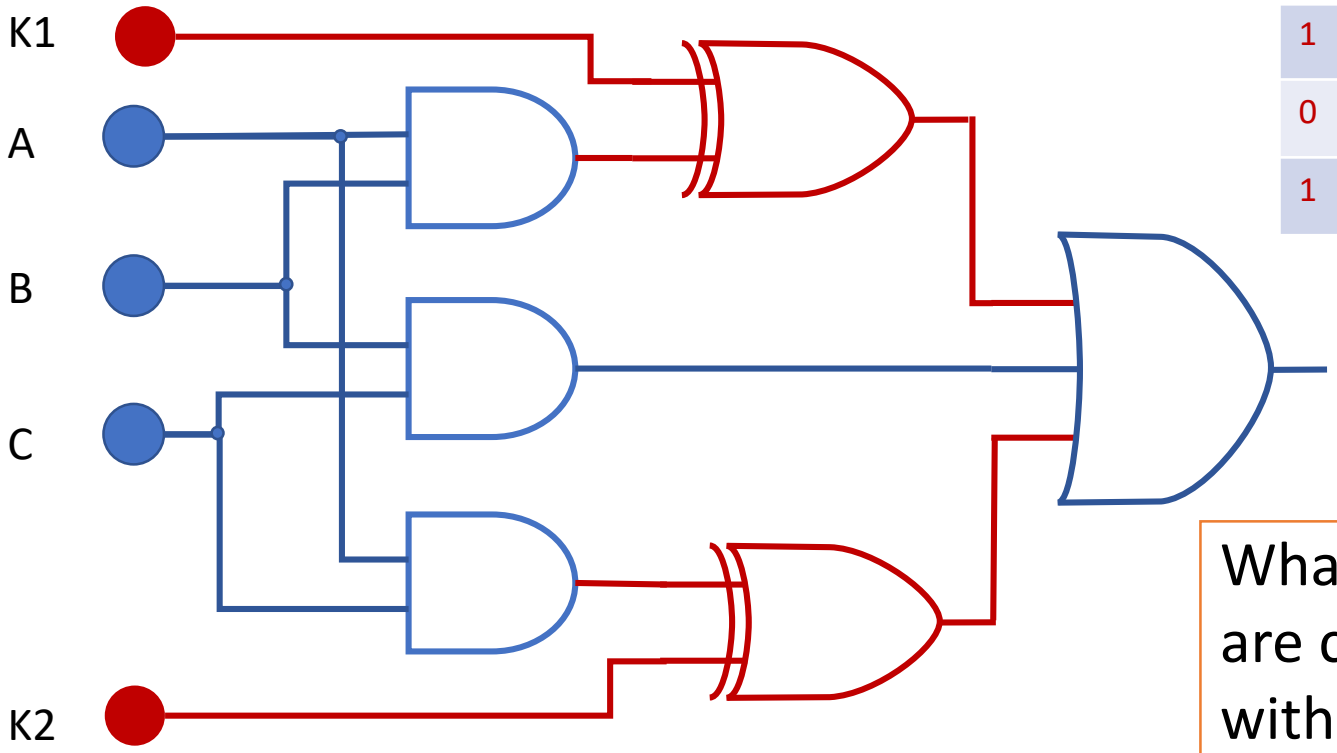
Suppose we have a locked circuit represented as C(X, K, Y)

- X: circuit inputs, K: key inputs, Y: outputs

**Strawman attack algorithm**

- Generate a lot of random inputs: $X_1, X_2, \ldots, X_N$
- Evaluate the output for each on activated IC: $Y_1, Y_2, \ldots, Y_N$
- Construct SAT formula: $C(X_1, K, Y_1) \wedge \cdots \wedge C(X_2, K, Y_2)$
- Satisfying assignment to $K$ is the key

# Strawman does not work




| A | B | C | Y |
|---|---|---|---|
| 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 |

What key values are consistent with the above I/O patterns?

(K1,K2)=(1,0); (K1,K2)=(0,0)

# Problems with the Strawman SAT Attack

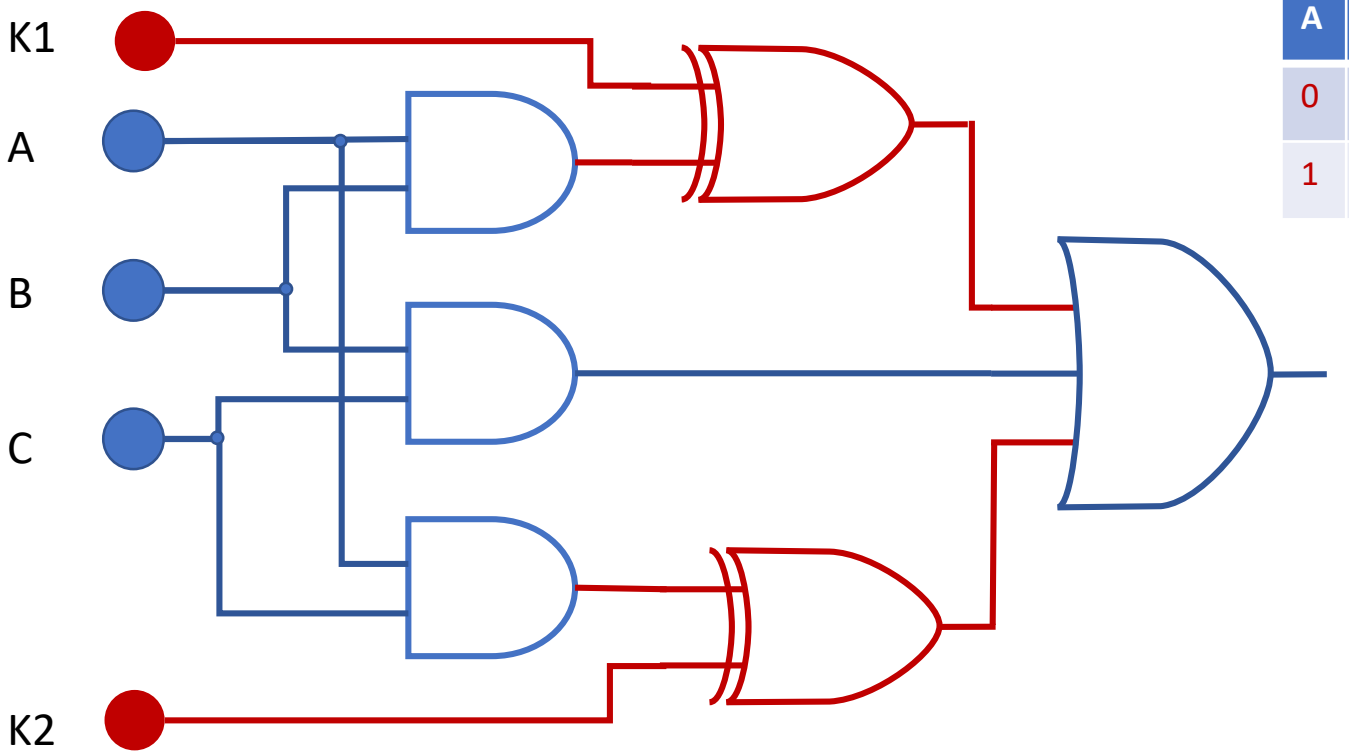
**Strawman attack algorithm**

- Generate a lot of random inputs: $X_1, X_2, \ldots, X_N$
- Evaluate the output for each on activated IC: $Y_1, Y_2, \ldots, Y_N$
- Construct SAT formula: $C(X_1, K, Y_1) \wedge \cdots \wedge C(X_2, K, Y_2)$
- Satisfying assignment to $K$ is the key

**Problems**

- Might have two keys which agree on $X_1, \ldots, X_N$ but differ on $X_{N+1}$
- So we don't know when to stop sampling inputs

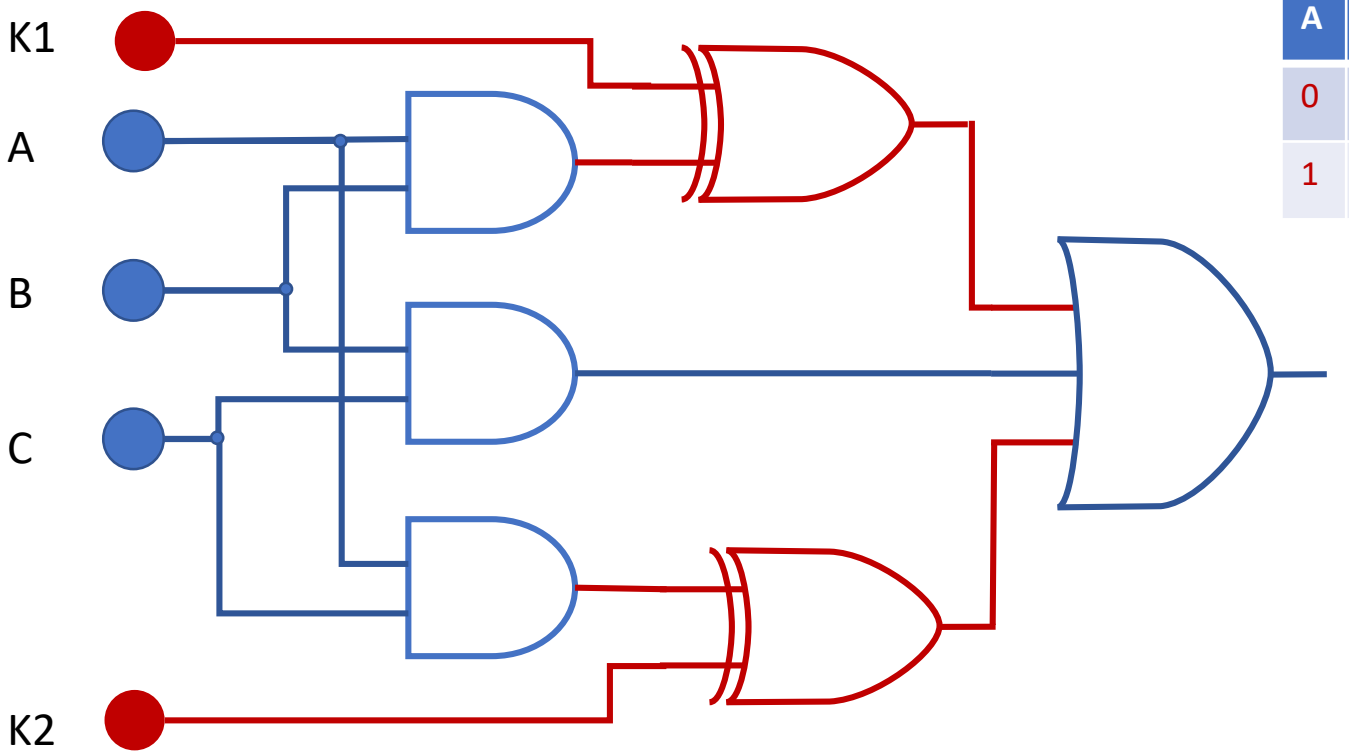
# Key Idea in SAT Attack: Distinguishing Inputs




| A | B | C | Y |
|---|---|---|---|
| 0 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 |

The above input vectors are useless because they result in the same output value for all keys

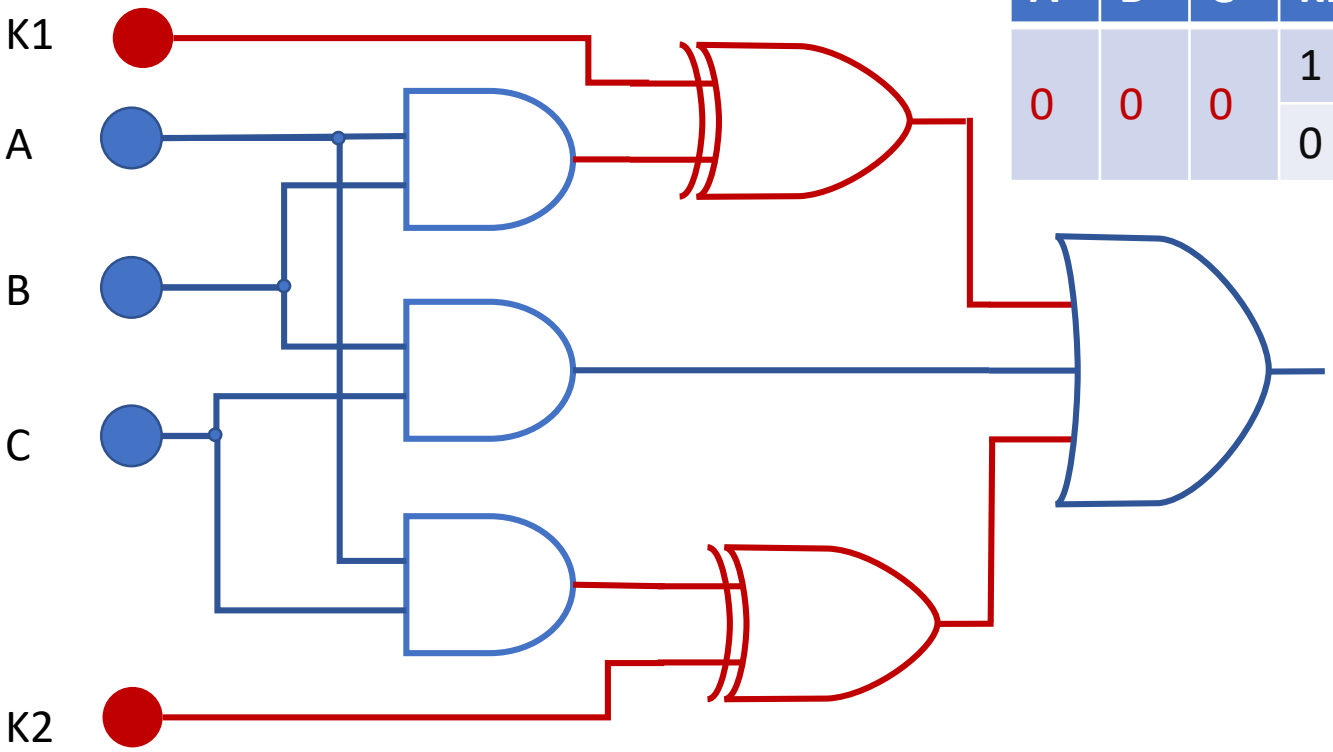
# Key Idea in SAT Attack: Distinguishing Inputs




| A | B | C | Y |
|---|---|---|---|
| 0 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 |

Want input vectors that can **distinguish** between at least two keys with **different behaviors**

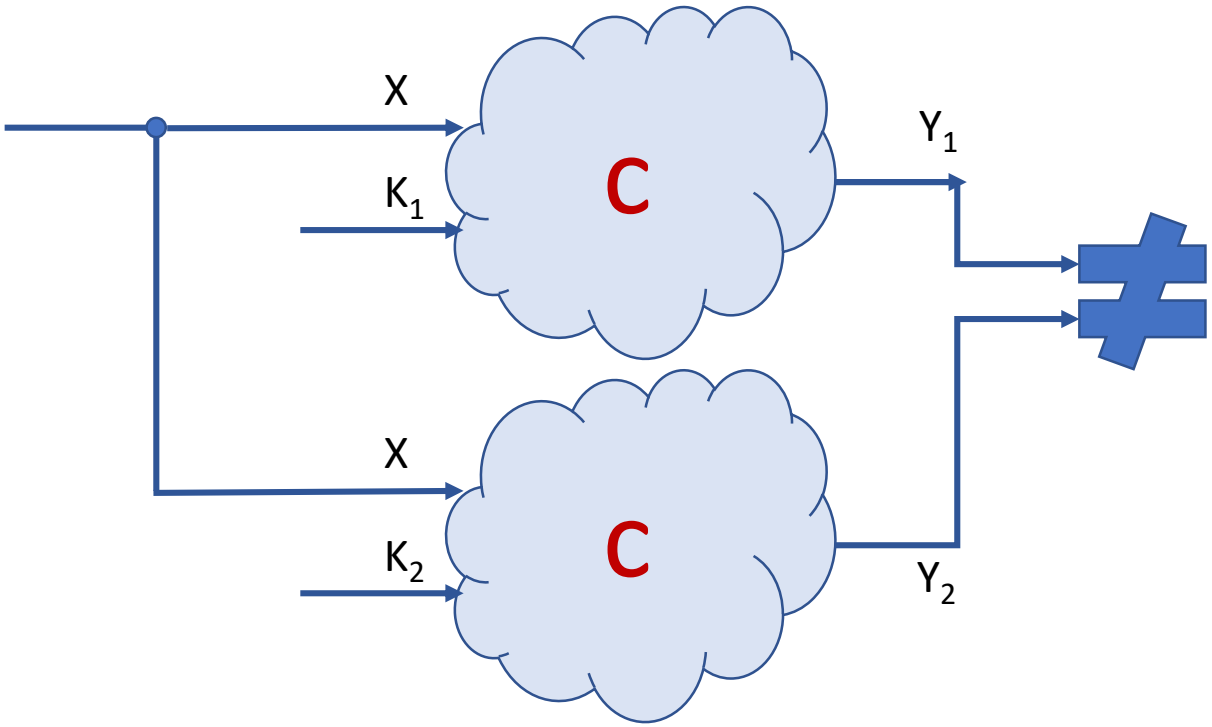
# Key Idea in SAT Attack: Distinguishing Inputs




| A | B | C | K1 | K2 | Y |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 1 | 1 |
| | | | 0 | 0 | 0 |

**Distinguishing input for two key values:** an input vector that produces different outputs for these keys

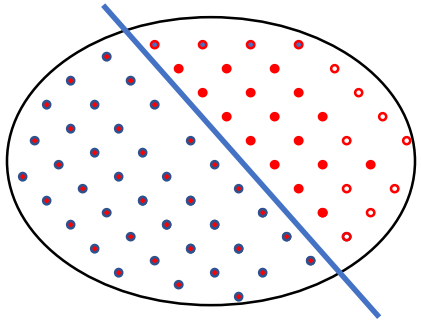
# How do we compute distinguishing inputs?



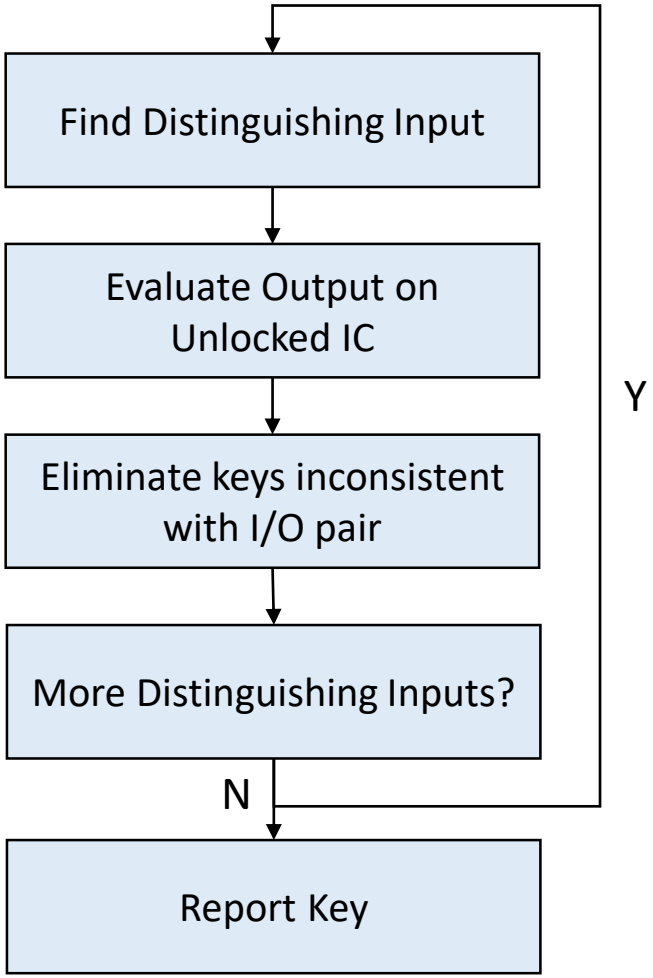

$$C(X, K_1, Y_1) \wedge C(X, K_2, Y_2) \wedge Y_1 \neq Y_2$$

# SAT Attack

I/O set E := {($i_1$,$o_1$)}

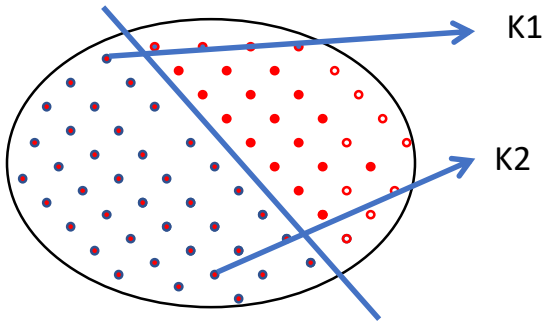


Space of all possible keys

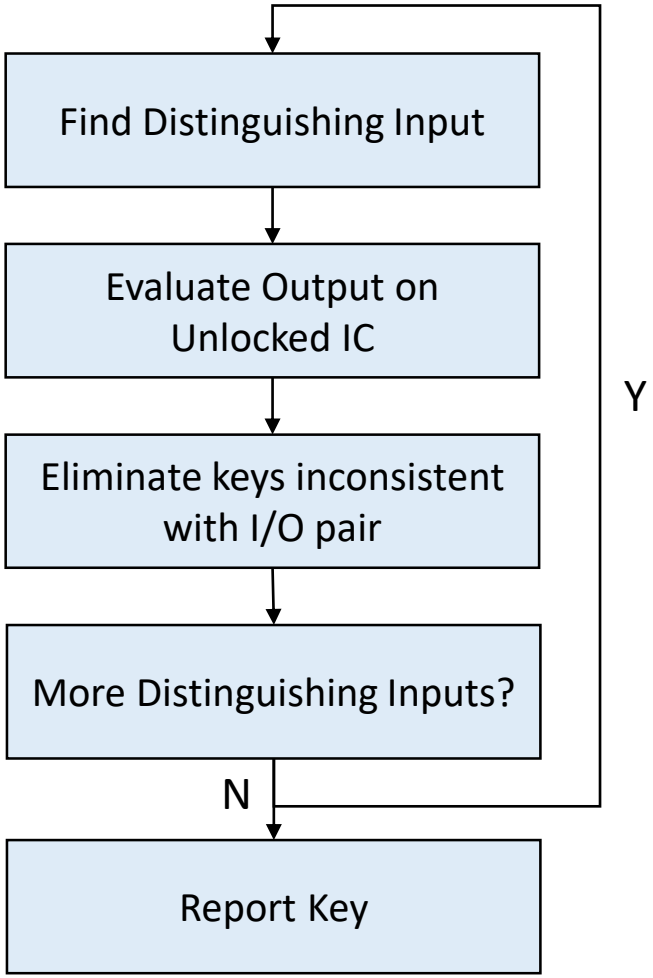


# SAT Attack

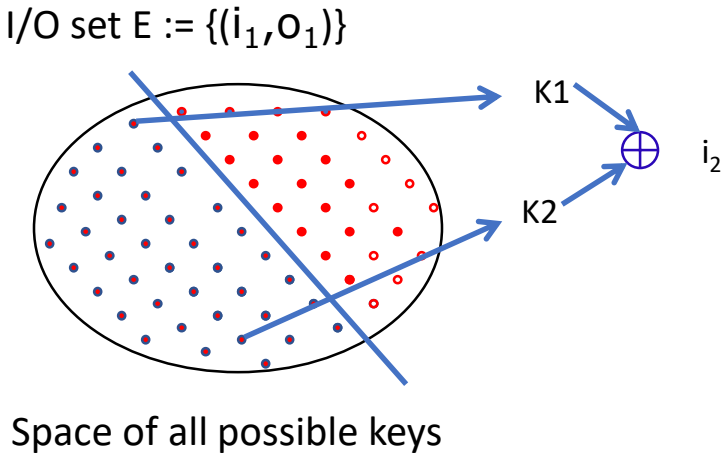
I/O set E := {($i_1$,$o_1$)}

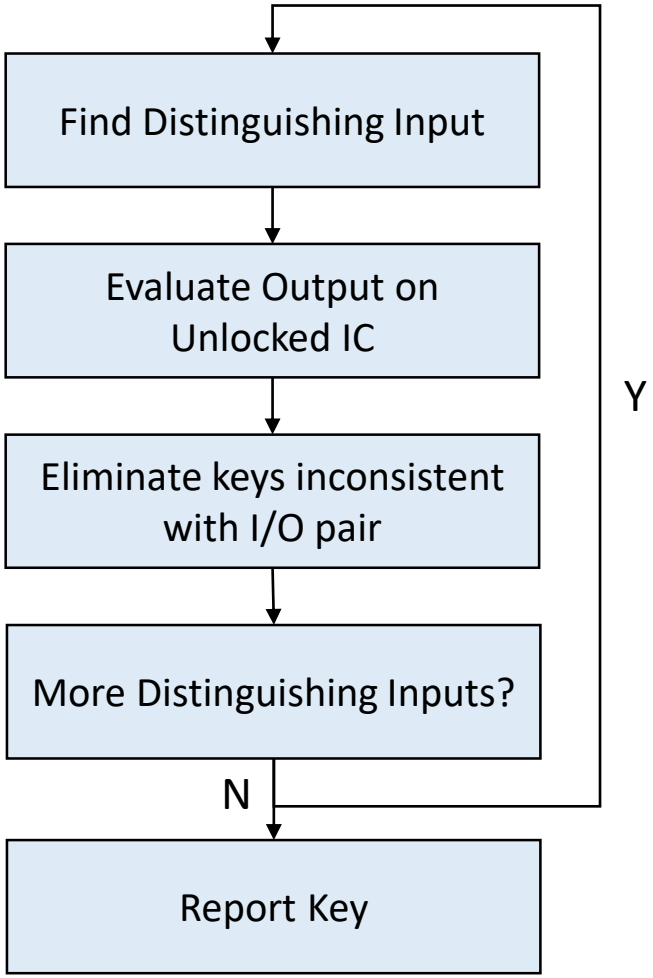



Space of all possible keys

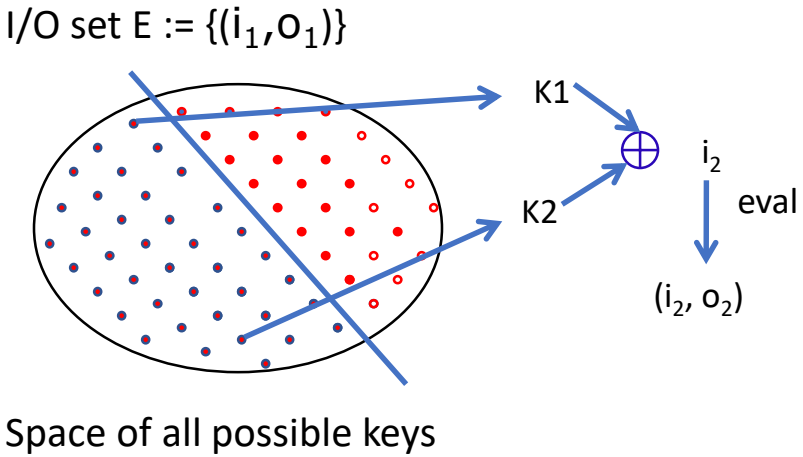


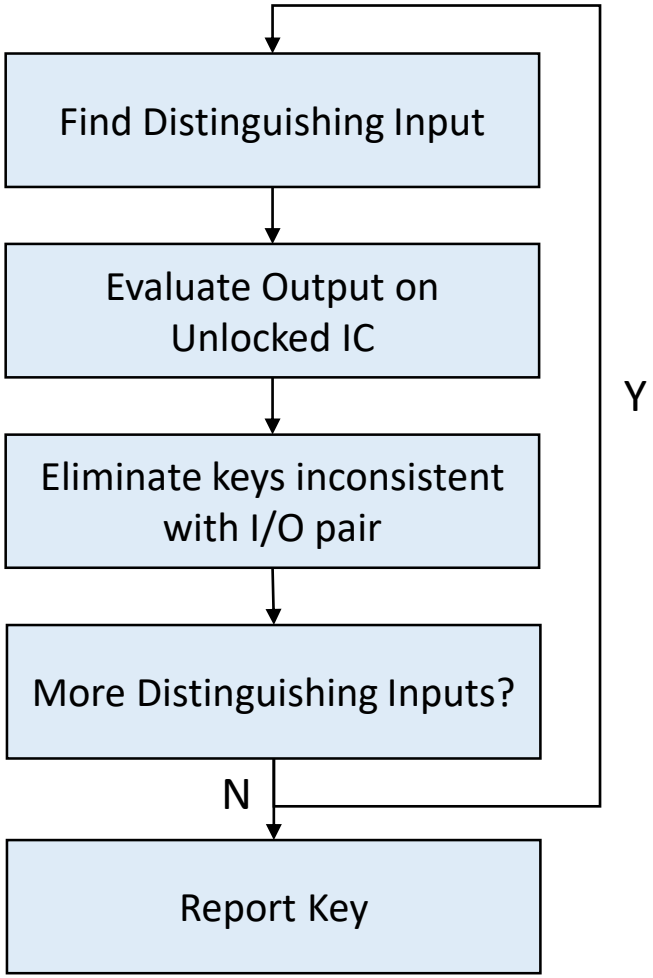
# SAT Attack


I/O set E := {($i_1$,$o_1$)}
K1
K2
$i_2$
Space of all possible keys


Find Distinguishing Input
Evaluate Output on Unlocked IC
Eliminate keys inconsistent with I/O pair
More Distinguishing Inputs?
N
Y
Report Key

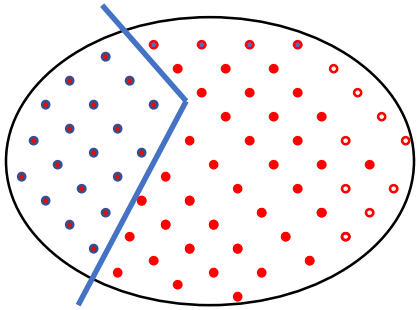
# SAT Attack


I/O set E := {($i_1$,$o_1$)}
K1
K2
$i_2$
eval
($i_2$, $o_2$)
Space of all possible keys

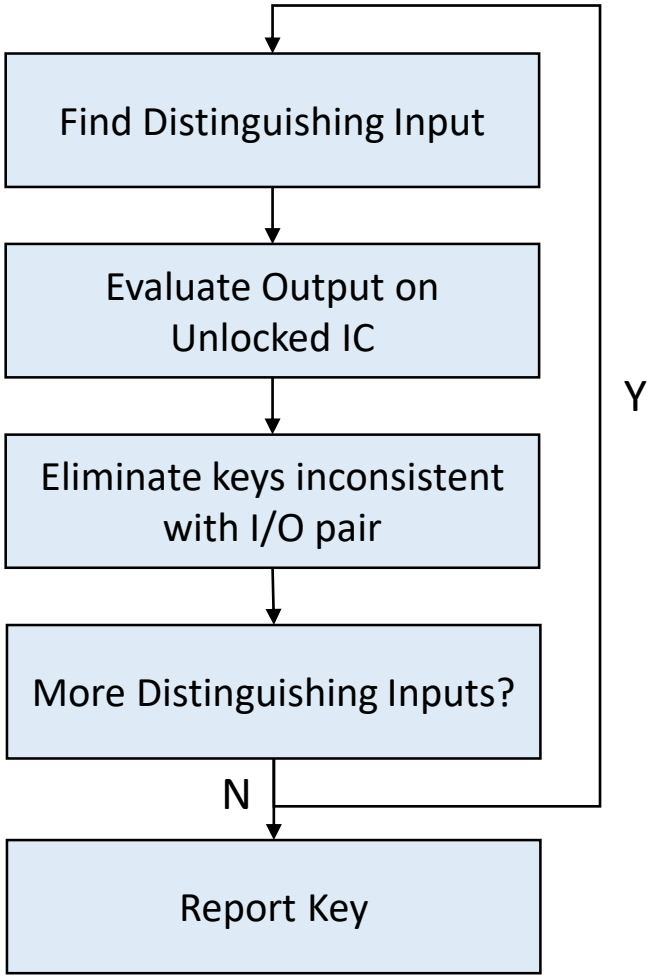

Find Distinguishing Input
Evaluate Output on Unlocked IC
Eliminate keys inconsistent with I/O pair
More Distinguishing Inputs?
Y
N
Report Key

# SAT Attack

I/O set E := E U $\{(i_2, o_2)\}$

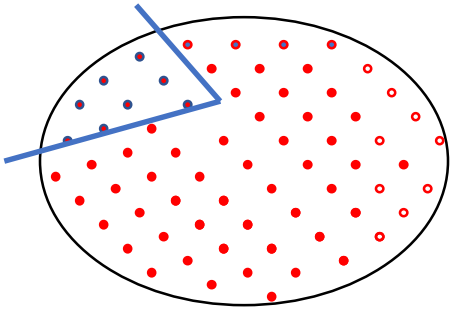


Space of all possible keys

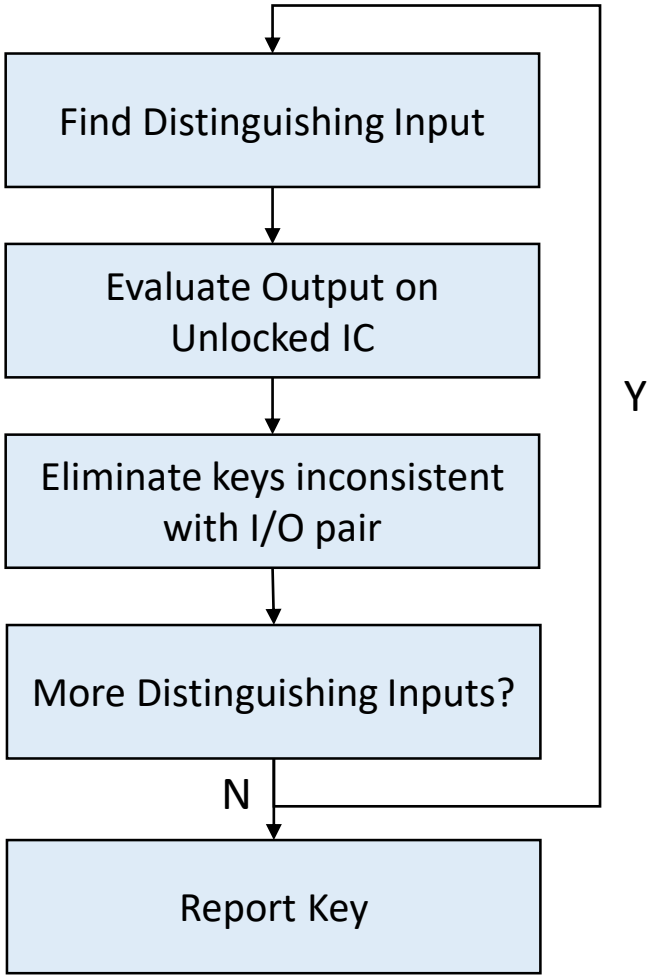


# SAT Attack

I/O set E := E ∪ {($i_j$,$o_j$)}

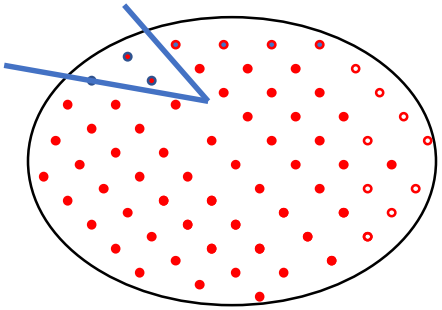


Space of all possible keys

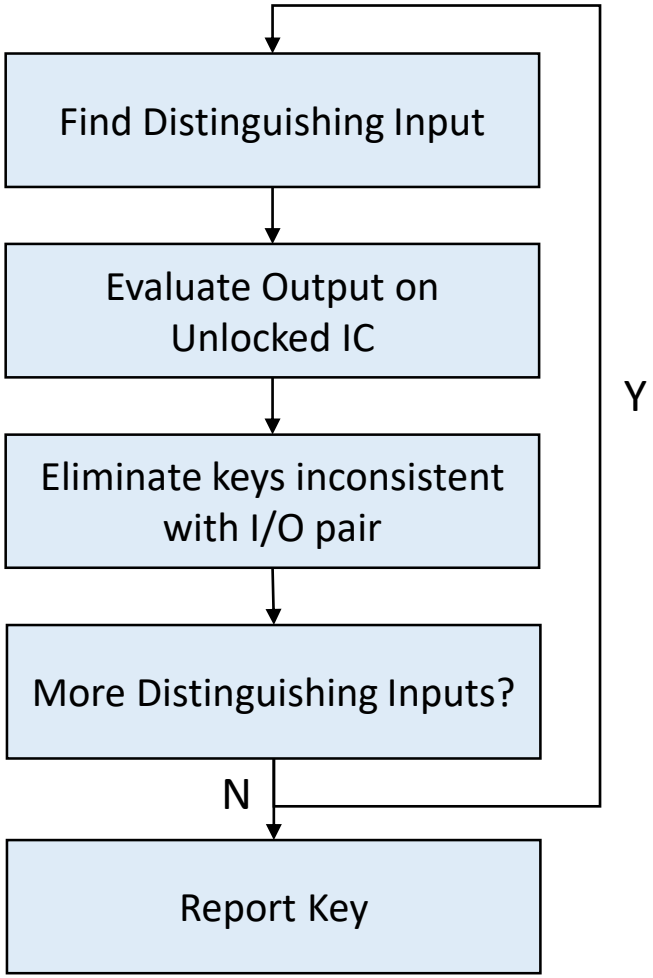


# SAT Attack

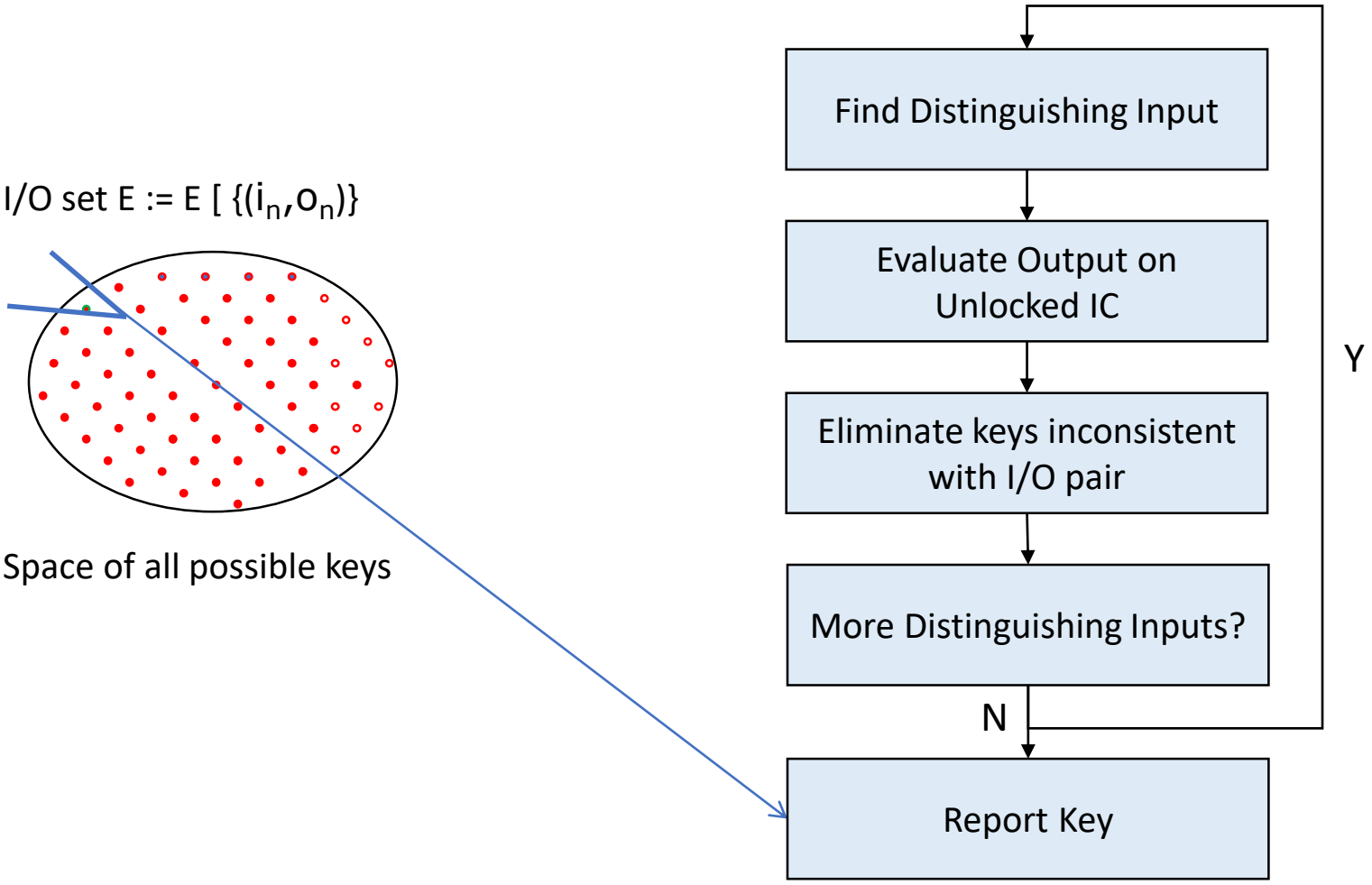
I/O set E := E ∪ {($i_k$,$o_k$)}



Space of all possible keys



# SAT Attack


I/O set E := E [ {(in,on)}
Space of all possible keys
Find Distinguishing Input
Evaluate Output on Unlocked IC
Eliminate keys inconsistent with I/O pair
More Distinguishing Inputs?
Y
N
Report Key

# SAT Attack: Algorithm

**function** SATAttack($C$)

$k \leftarrow 1$

$R_1 \leftarrow C(X, K_1, Y_1) \wedge C(X, K_2, Y_2) \wedge (\theta \leftrightarrow Y_1 \neq Y_2)$

**while** $sat(R_k \wedge \theta)$ **do**

$\Delta \leftarrow \text{MODEL}_{(X)}(R_k \wedge \theta)$

$O \leftarrow Eval_i(\Delta)$

$O_1 \leftarrow C(\Delta, K_1, O)$

$O_2 \leftarrow C(\Delta, K_2, O)$

$R_{k+1} \leftarrow R_k \wedge O_1 \wedge O_2$

$k \leftarrow k + 1$

**end while**

**if** $sat(R_k \wedge \neg\theta)$ **then**

**return** $\text{MODEL}_{\text{K1}}(R_k \wedge \neg\theta)$

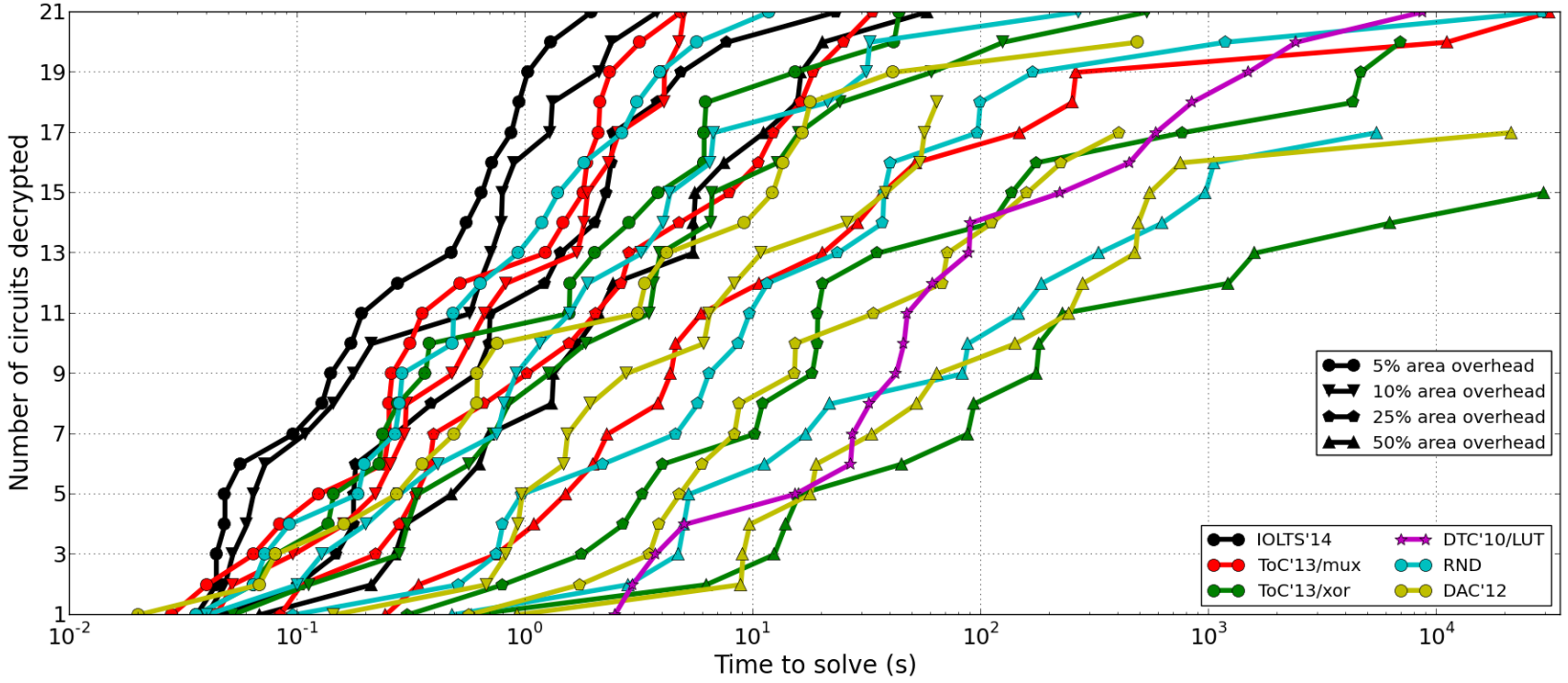
**end if**

**return** $\perp$

**end function**

# SAT Attack Results

[Subramanyan et al., HOST'15]



# SAT Attack Impact

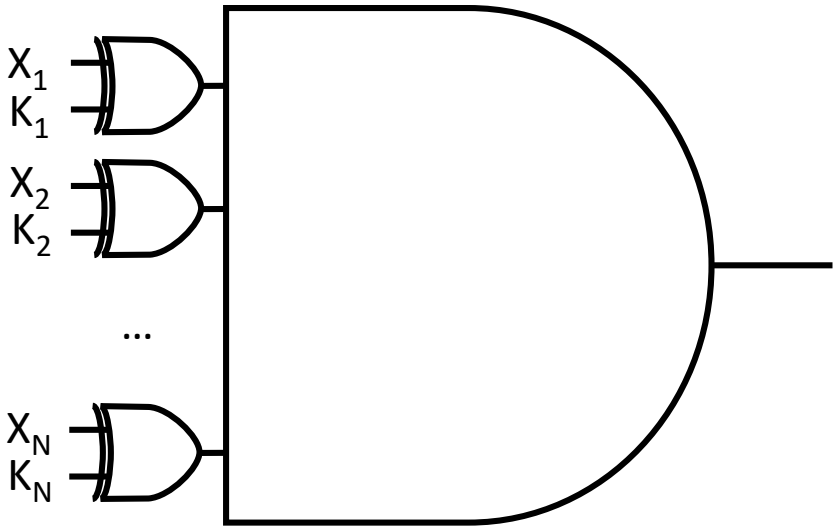
[Subramanyan et al., HOST'15]

- Broke all combinational locking methods known at the time
- Led to a spate of new papers on SAT attack resilient locking

# SAT Resilient Locking Methods

# The curious case of c2670 [Subramanyan et al., HOST'15]

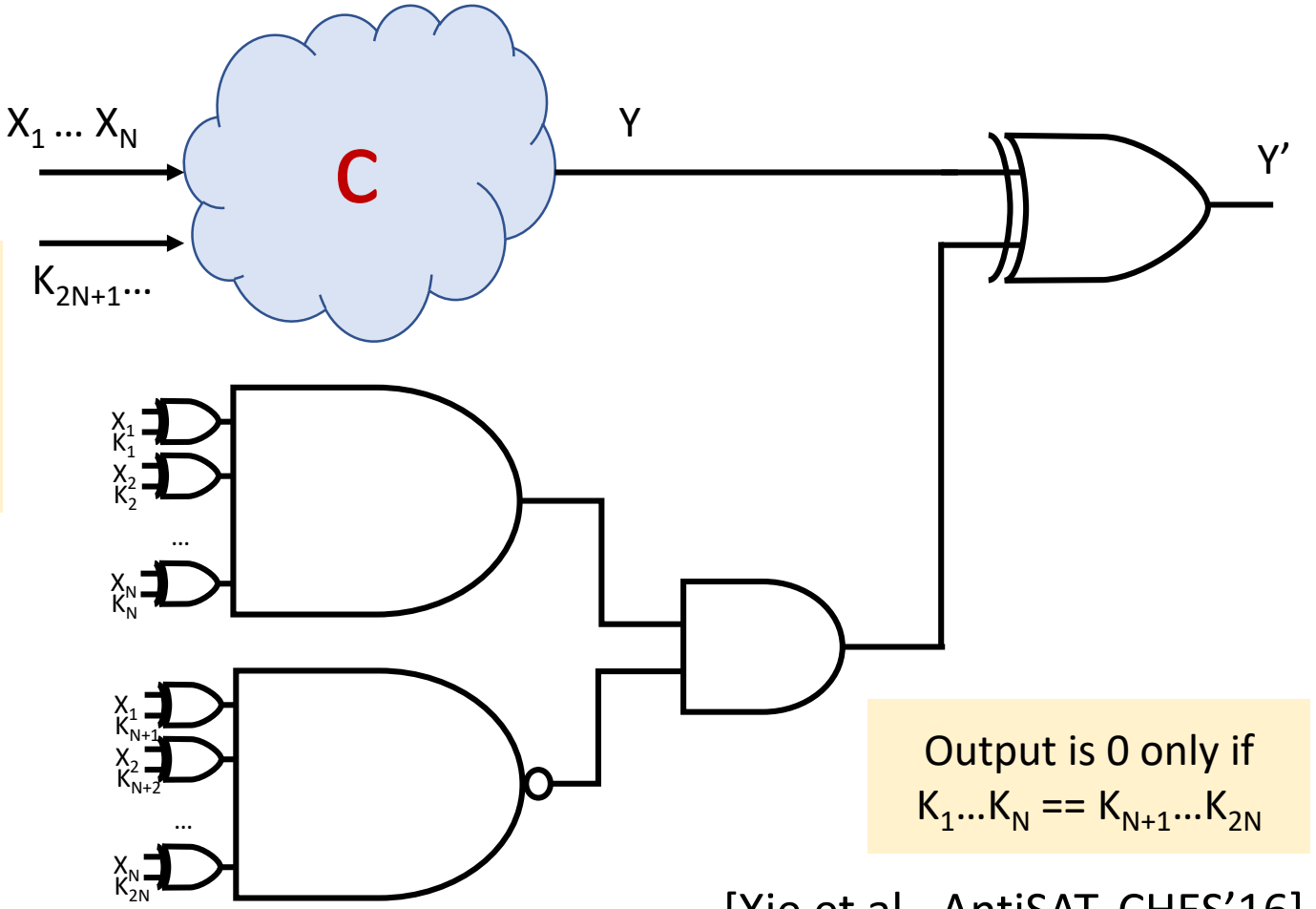
- Most resilient circuit to SAT attack in experiments
- Note benchmark set included DES, multipliers, etc.
- So why was this small circuit the most difficult?




This circuit requires $2^{N/2}$ distinguishing inputs on average to find the key

# Circuit+AND Tree ⇨ SAT Resilience

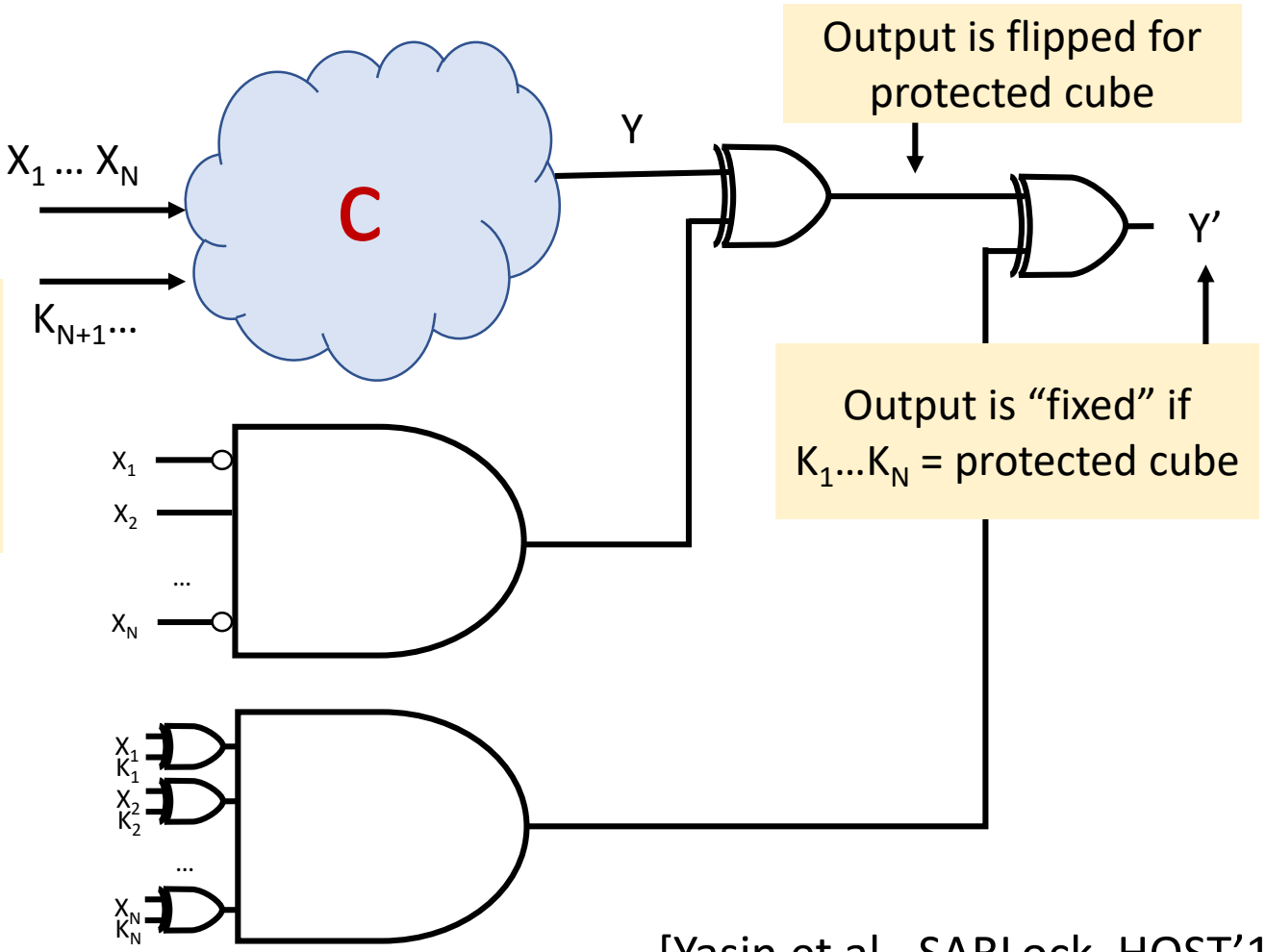
Use pre-SAT locking here for output corruptibility




Output is 0 only if $K_1 \ldots K_N == K_{N+1} \ldots K_{2N}$

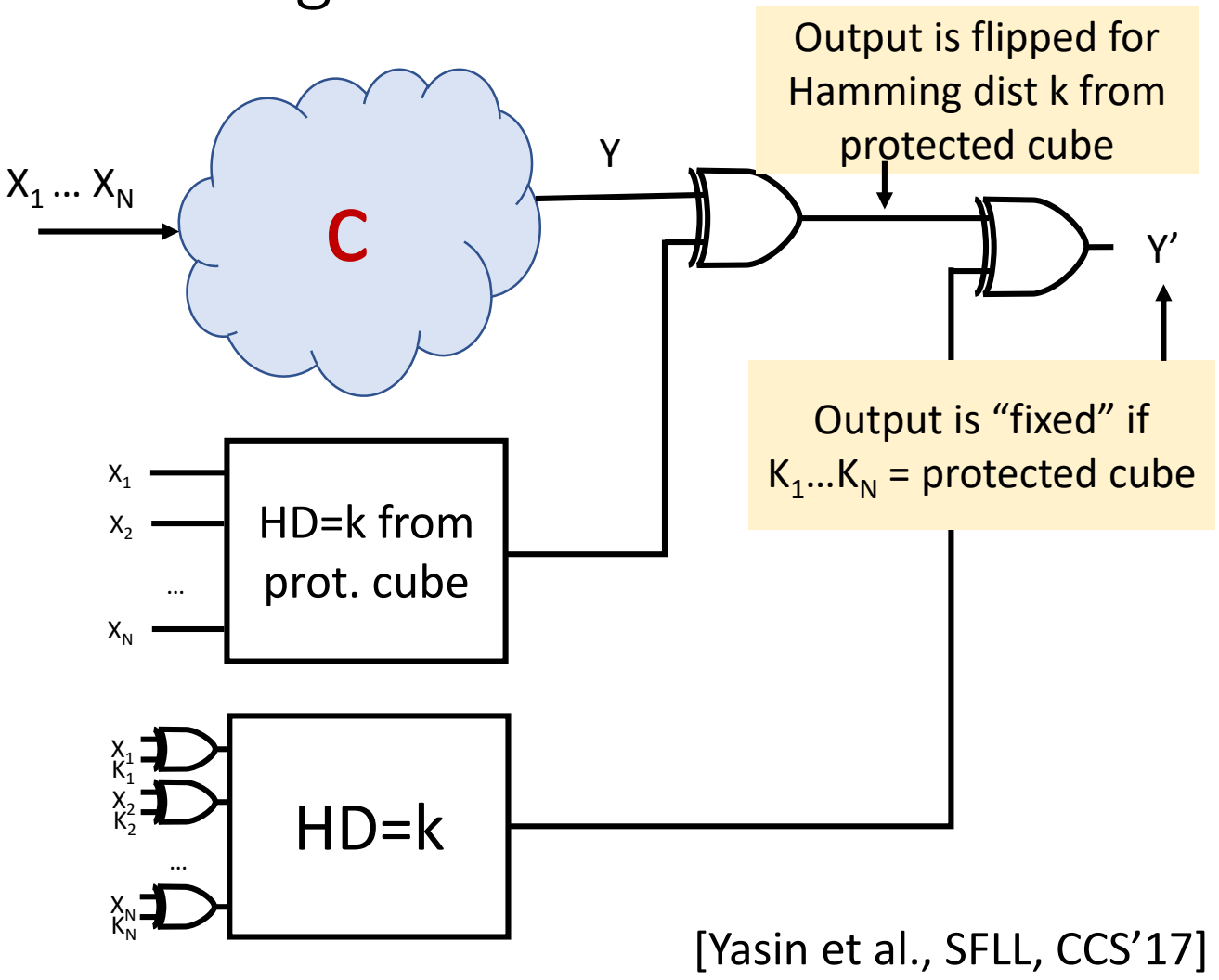
[Xie et al., AntiSAT, CHES'16]

# Circuit+AND Tree ⇨ SAT Resilience



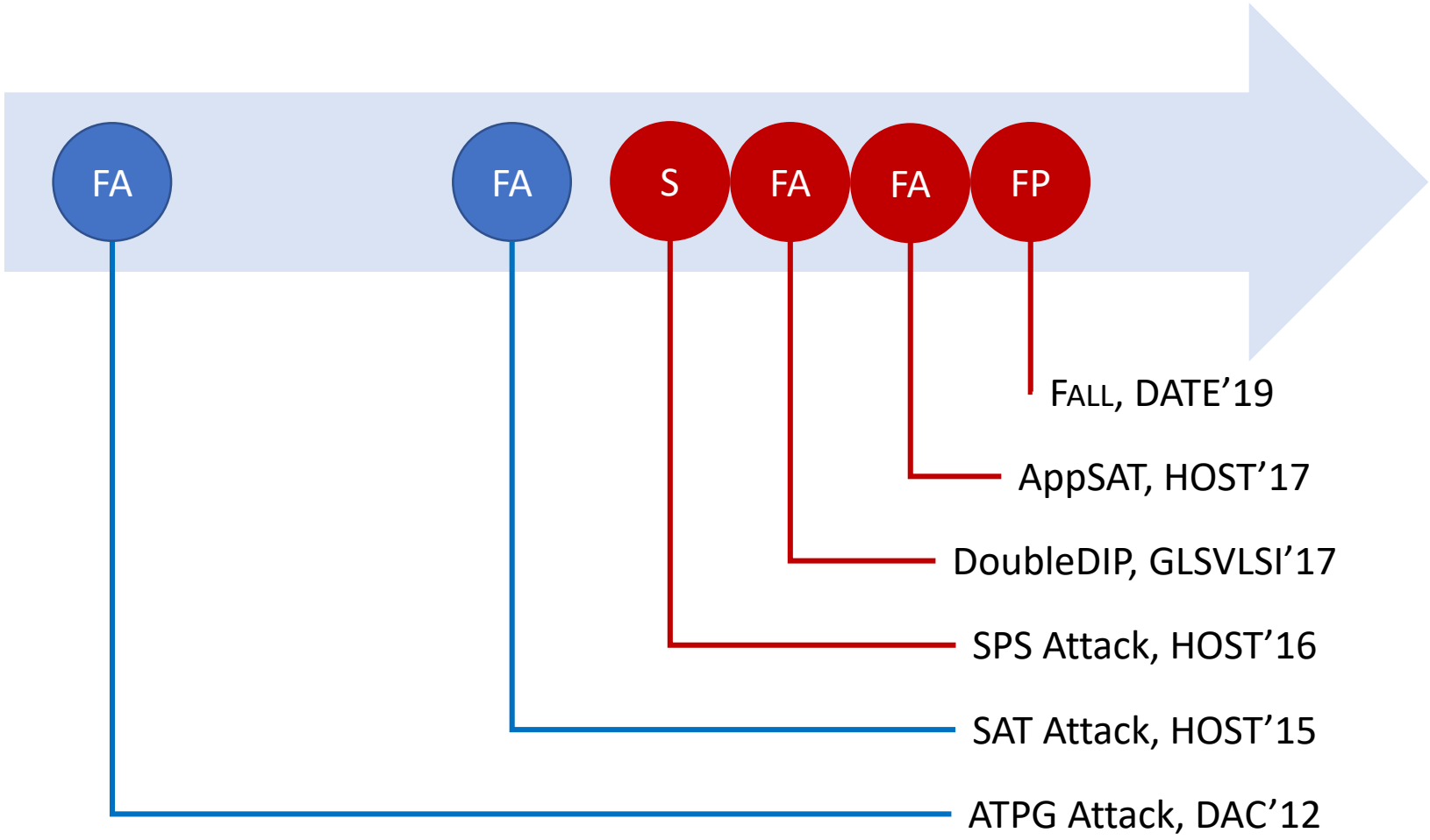

[Yasin et al., SARLock, HOST'16]

# Circuit+Hamming Distance⇨ SAT Resilience




[Yasin et al., SFLL, CCS'17]

# Subsequent Attacks



# Signal Probability Skew Attack [Yasin et al., HOST'16]

Key idea: output signal S of AND-tree has very low Pr(S = 1)

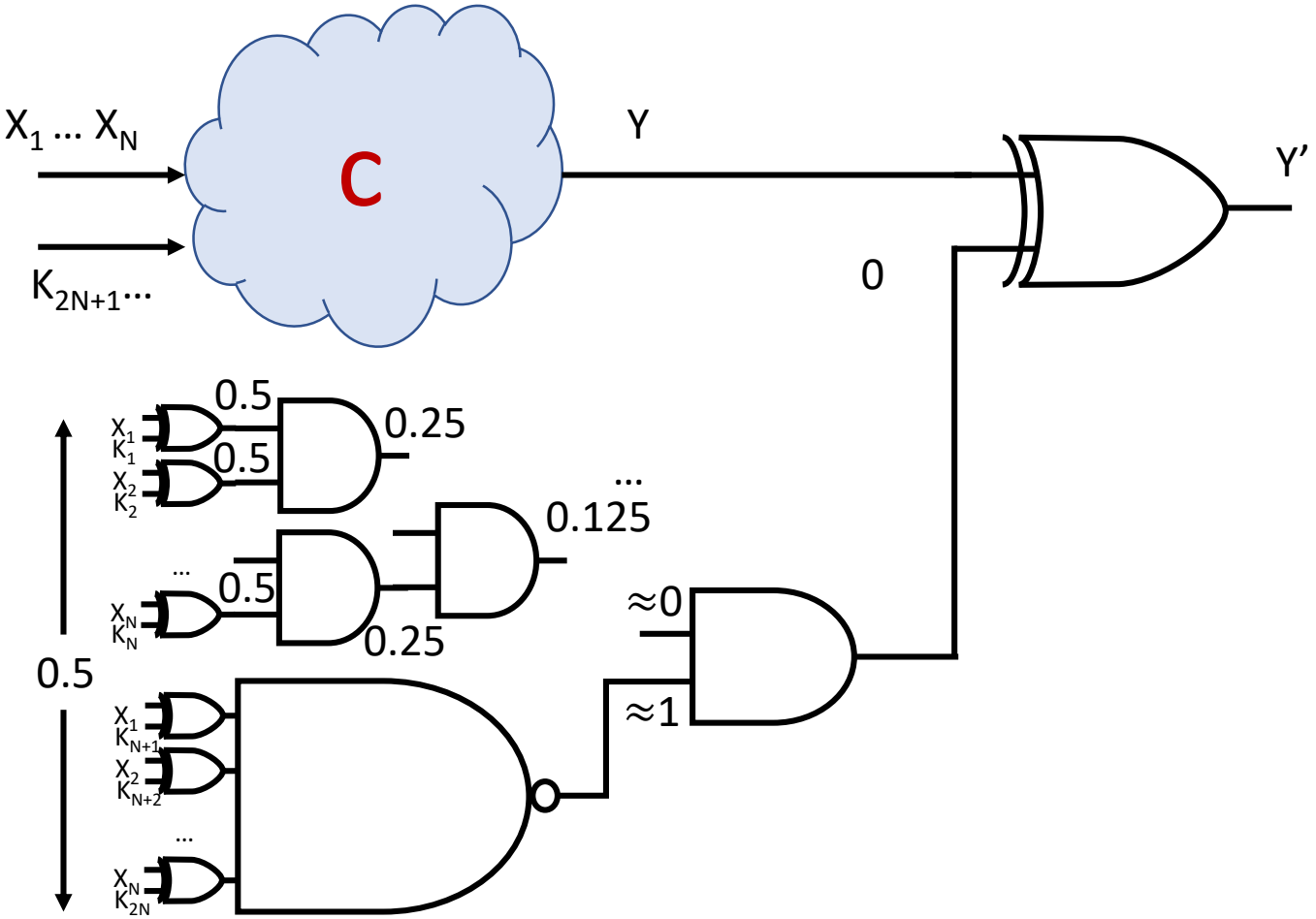
So isolate such nodes and **remove them**

How to isolate?

- Set $Pr(X_i = 1) = 0.5$ for all primary inputs
- For AND gate, Y = AND(A, B)
  - Pr(Y=1) = Pr(A=1) * Pr(B=1)
- For OR gate, Y = OR(A,B)
  - Pr(Y=1) = (1 – Pr(A=1)) * (1 – Pr(B=1))
- And so on …

# SPS on Anti-SAT

[Yasin et al., HOST'16]



# DoubleDIP

[Shen and Zhou, GLSVLSI'16]

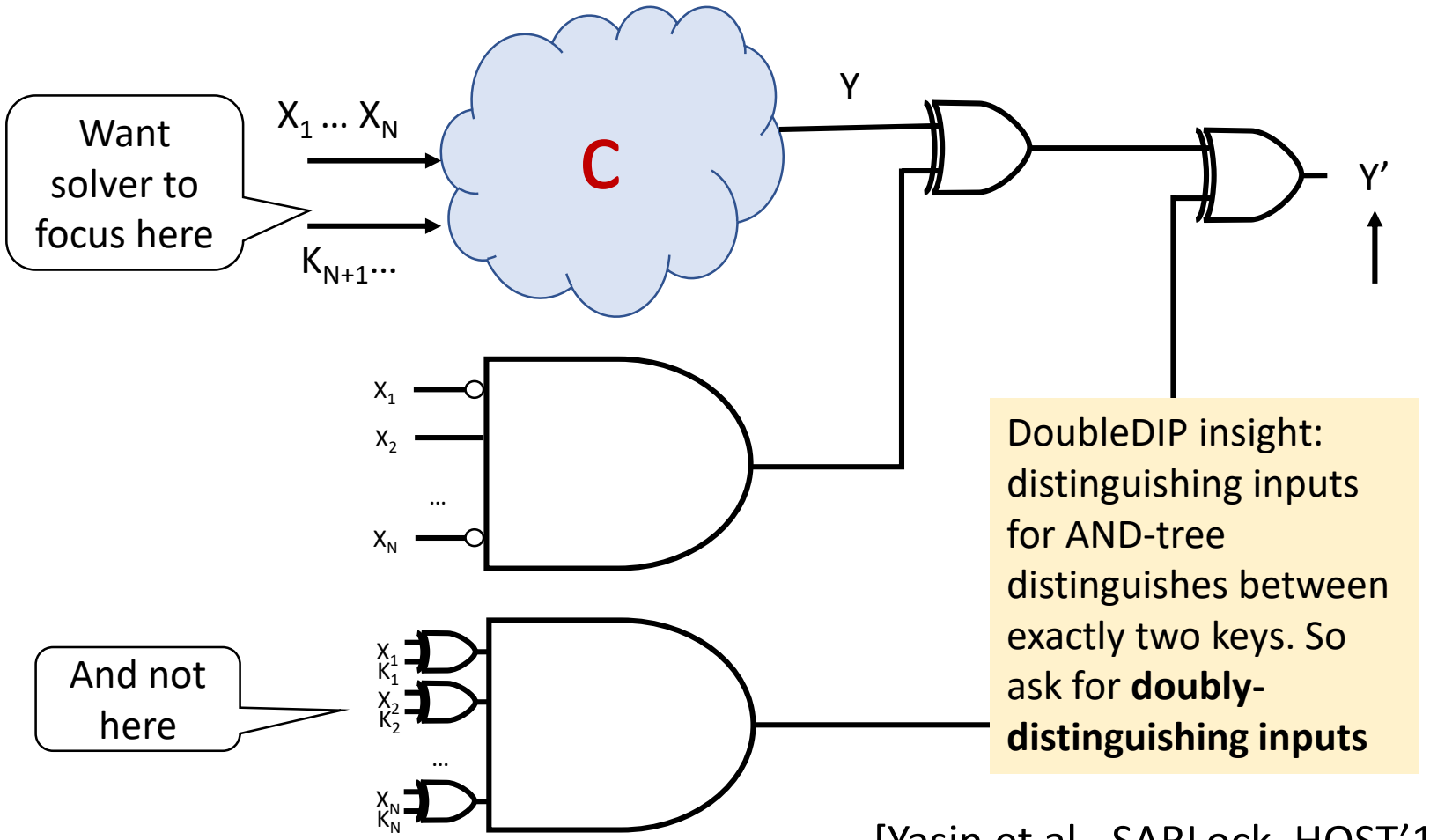
SARLock without the output corruptibility part produces the wrong output on exactly two input vectors

- We want to attack the output corruptibility inducing part of SARLock
- Left to itself, the SAT attack gets lost attacking the AND-tree
- **Double-DIP idea**: ask SAT solver to find doubly distinguishing input

$$C(X,K_1,Y_1) \wedge C(X,K_2,Y_2) \wedge C(X,K_3,Y_1) \wedge C(X,K_4,Y_2) \wedge Y_1 \neq Y_2 \wedge K_1 \neq K_3 \wedge K_2 \neq K_4$$

# DoubleDIP on SARLock

[Shen and Zhou, GLSVLSI'16]




[Yasin et al., SARLock, HOST'16]

# AppSAT

[Shamsi et al., HOST'17]

- Also focuses on lack of output corruptibility in SARLock
- Idea: run the SAT attack for a while, extract a key, and sample I/O behavior of this key. If the sampled I/O patterns seem alright, we can terminate even if the solver is able to find more distinguishing inputs
- This works pretty well against SARLock

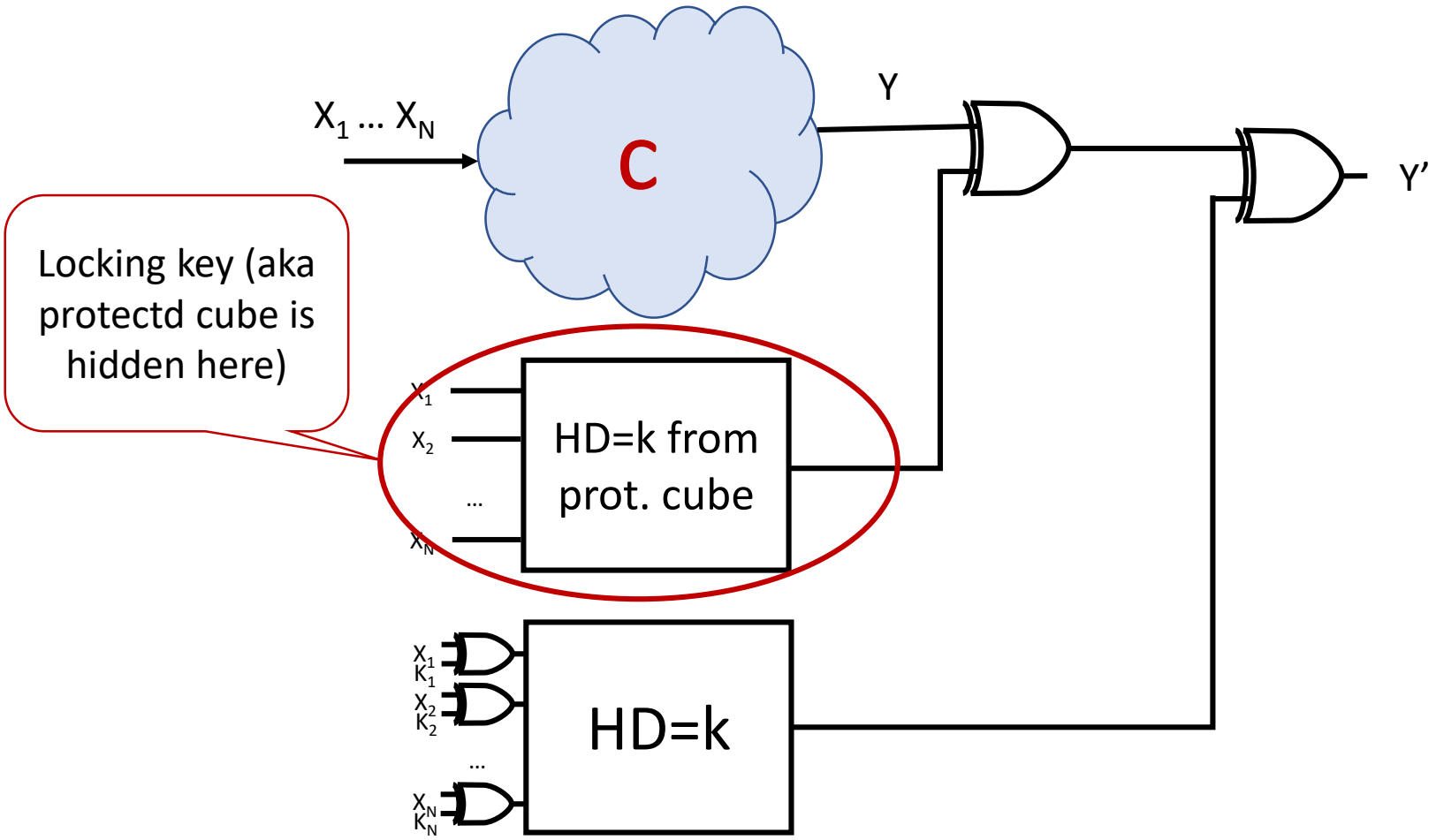
# FALL Attacks

[Sirone et al., DATE'19]

- Hamming Distance module has very specific Boolean properties
- This can be used to identify it and the protected cube
- This defeats SFLL and brings us back where we were in May 2015
- No published locking algorithm that is known to be secure exists

# FALL Attacks on SFLL

[Sirone et al., DATE'19]

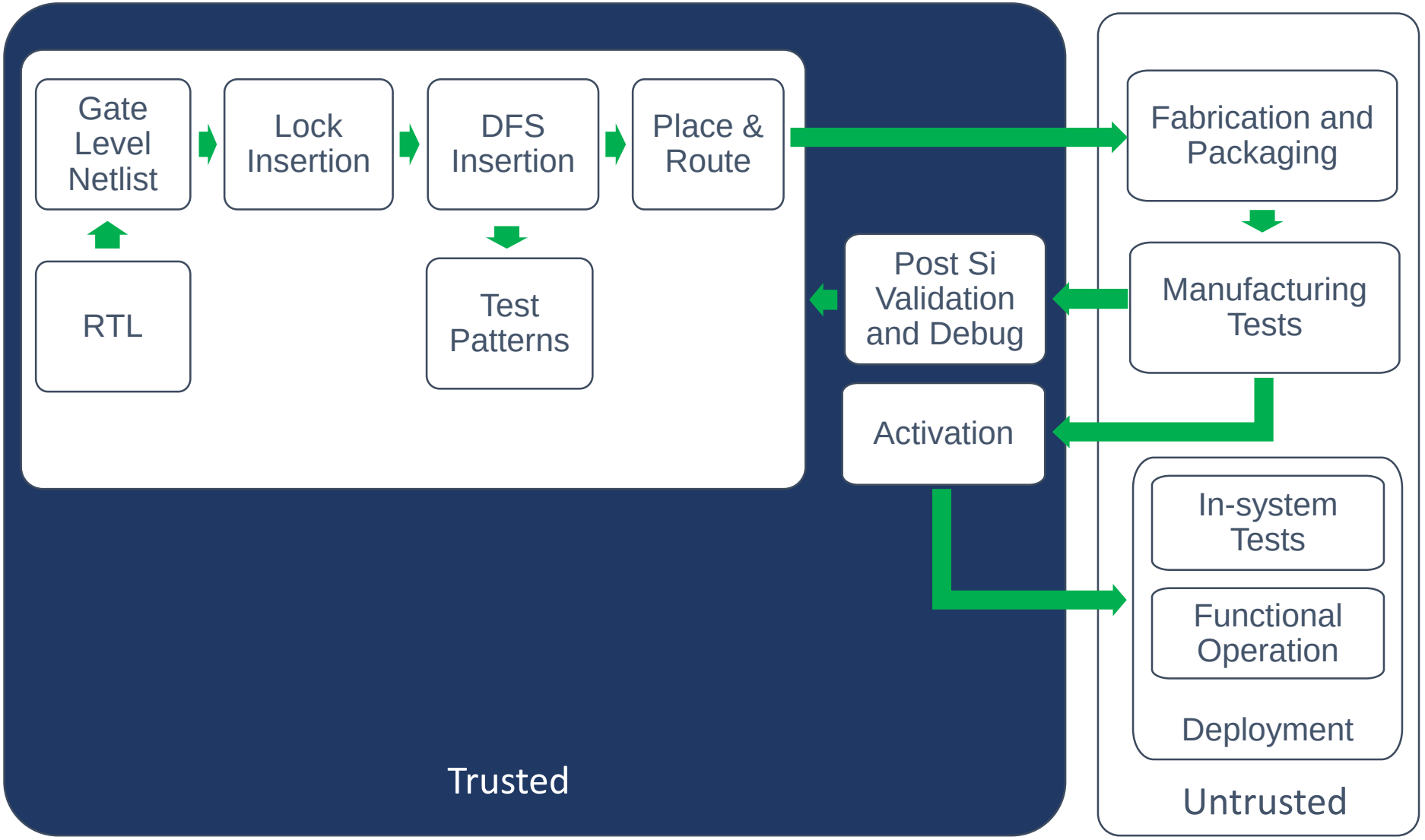


# Part III: Countermeasures

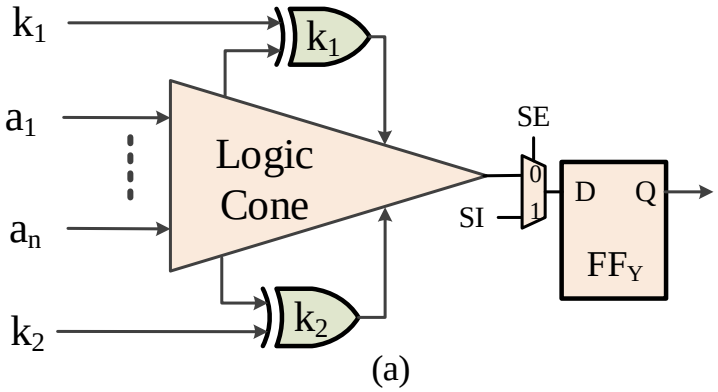
# Countermeasure 1: Design-for-Security (DFS) Architecture

- Design-for-security (DFS) architecture
  - Allows a fully scan-based structural tests at the foundry on the obfuscated design.
  - Once the keys programmed, and shipped to the customer, the scan out capability is blocked. Only scan-in and functional operation are supported. Full structural scan tests are still possible.
- DFS architecture prevents leaking of key information (even in part) to an adversary under any circumstances.

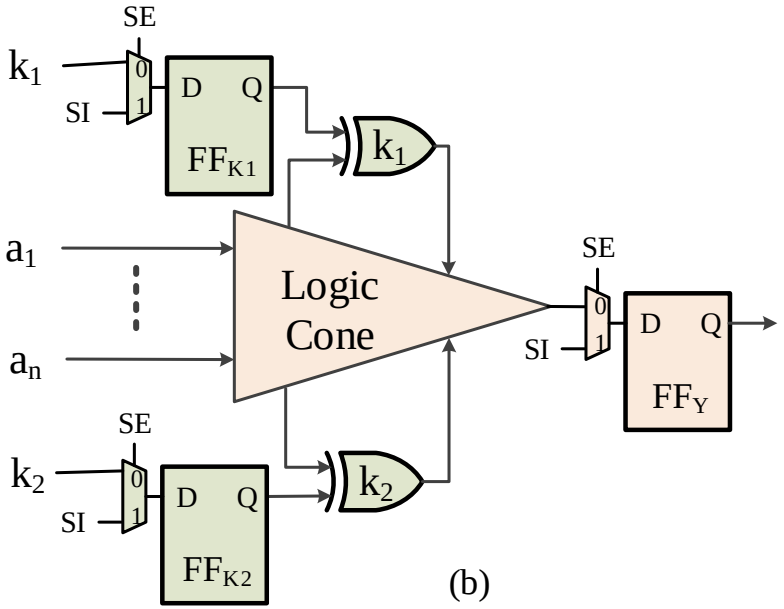
# Countermeasure 1: DFS Arch – Cont.



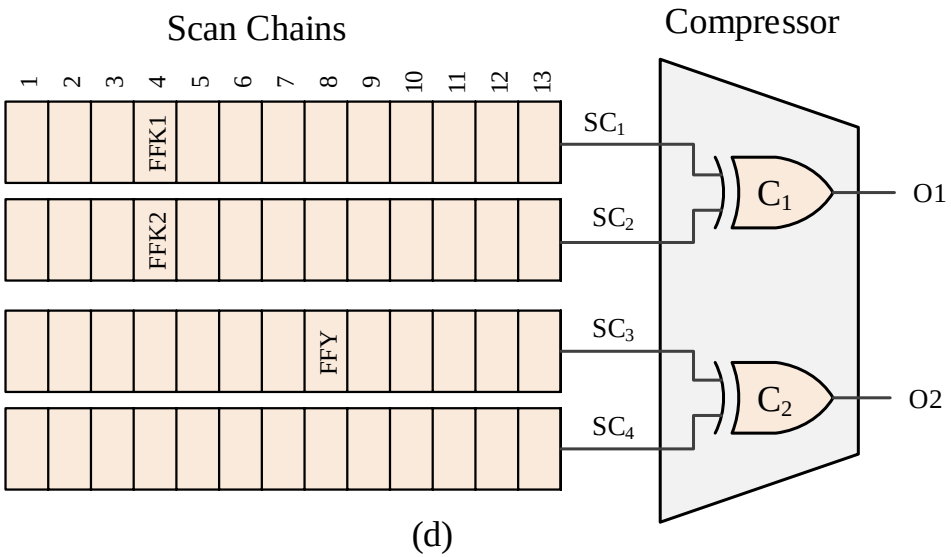
# Countermeasure 1: DFS Arch – Cont.



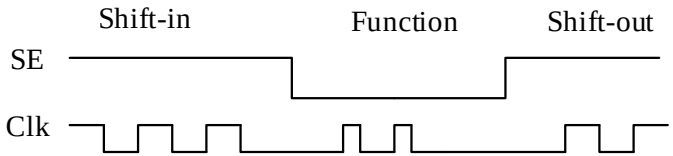
(a)



(b)

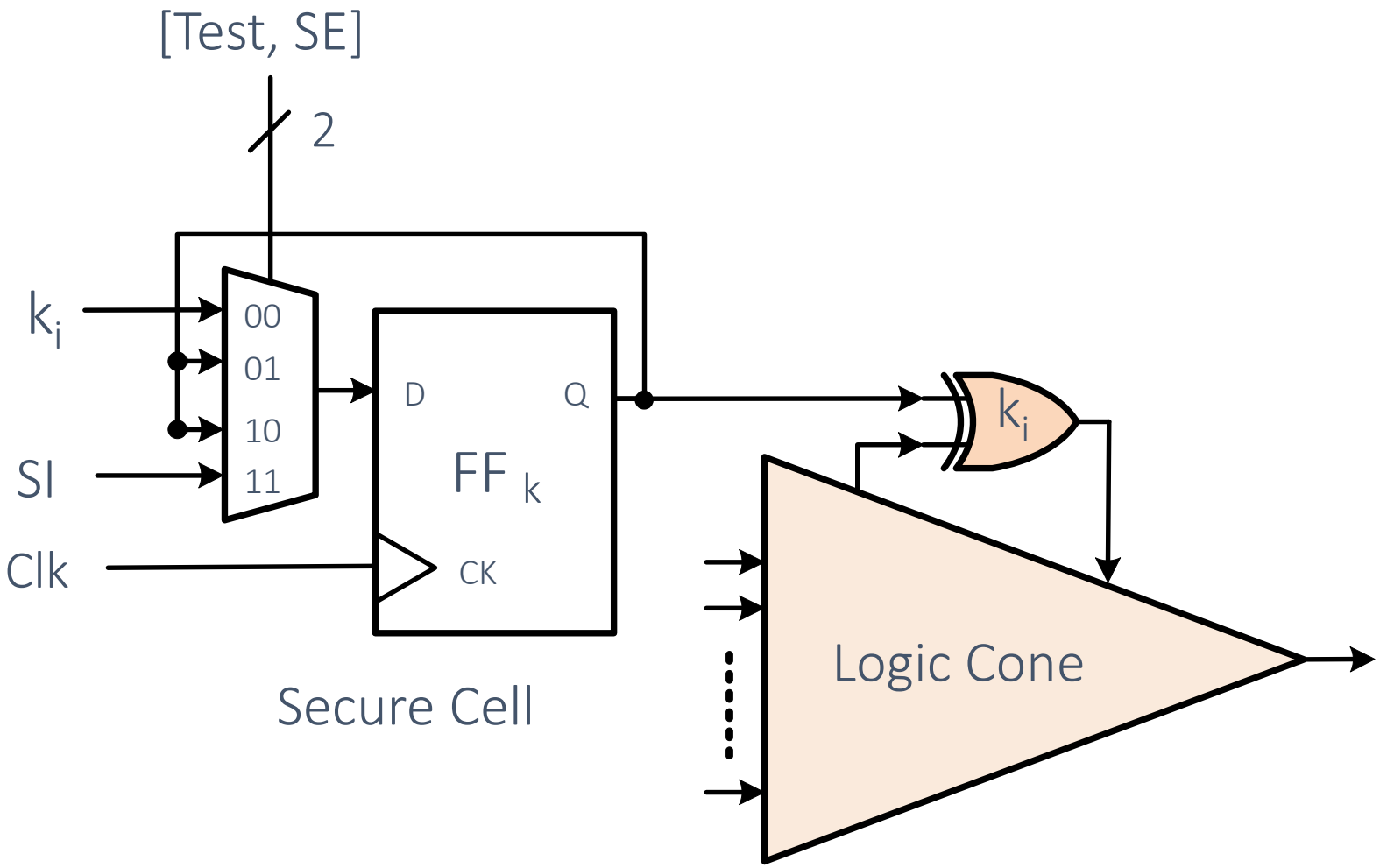


(d)

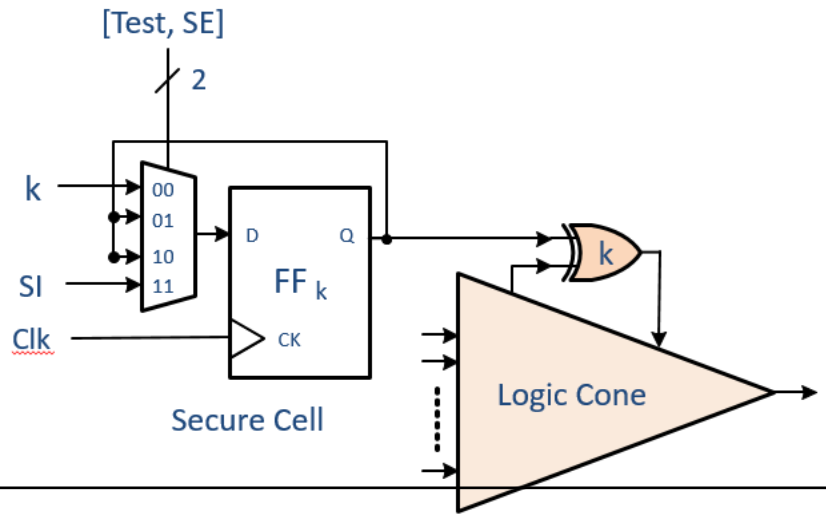


(c)

# Countermeasure 1: DFS Arch – Cont.



# Modes of Operation

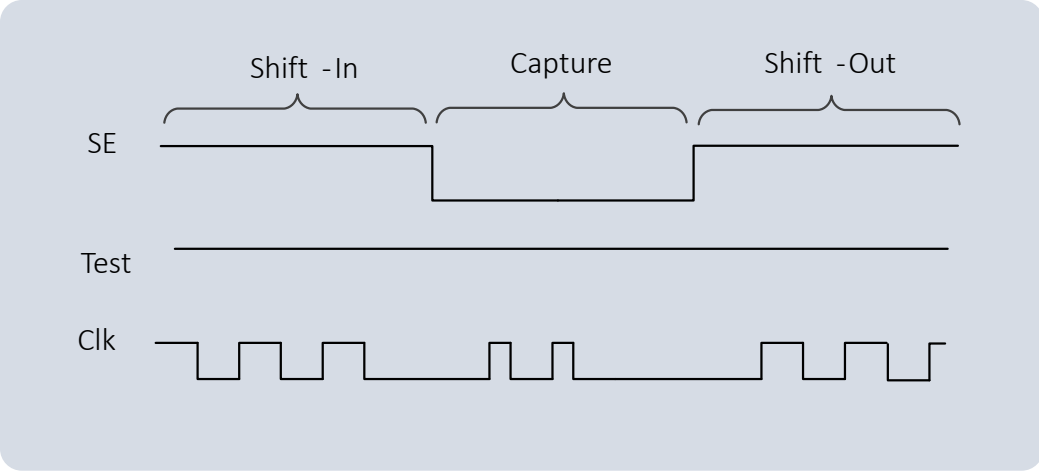



| Test | SE | Mode | Description |
|---|---|---|---|
| 0 | 0 | M0 | The chip is in functional mode. The secure cell applies key to the logic. |
| 0 | 1 | M1 | The secure cell holds its previous value. The rest of the circuit is in functional/ shift mode depending on the SE. |
| 1 | 0 | | |
| 1 | 1 | M2 | The SC becomes scan cell and it becomes a part of the scan chain. |

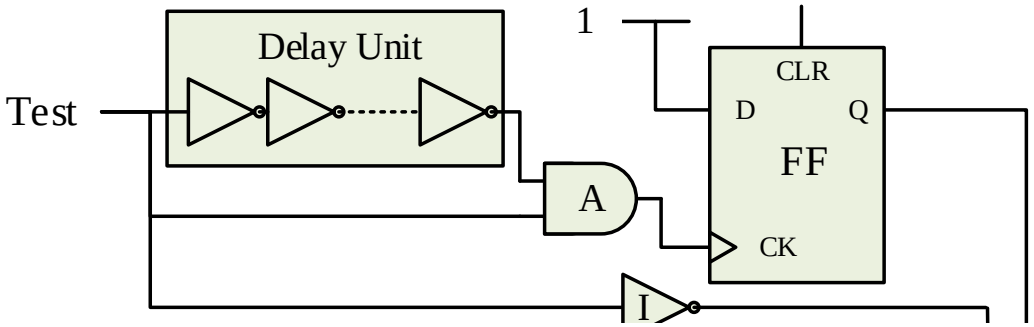
# Modes of Operation- Cont.

- Manufacturing Tests
  - No key required
- Post-Si Debug and Validation
  - Key is shifted through the scan chain
  - Performed at the trusted site
- In-system Tests
  - Disable scan

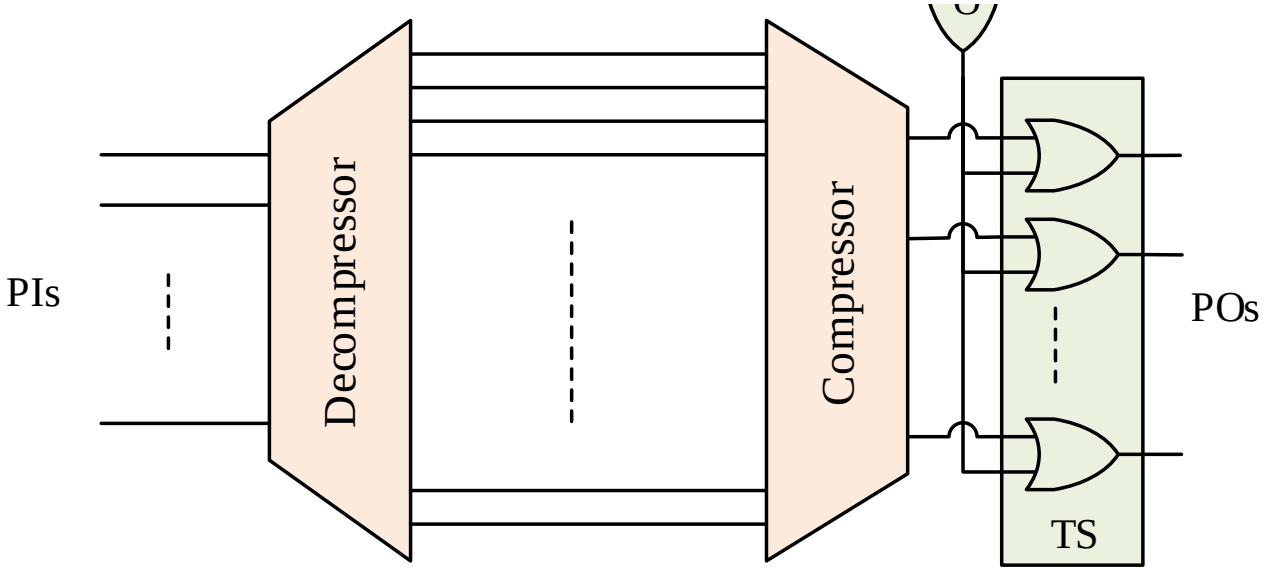
  dump



# Scan Data Access Control




No positive transition at the Test pin



# Results

TEST METRIC COMPARISON

| Benchmark | Key Bits | Test Coverage (%) | | | | Pattern Count | | | | Area Overhead |
|---|---|---|---|---|---|---|---|---|---|---|
| | | ORG | KEY | DFS | Change | ORG | KEY | DFS | Change(%) | |
| S35932 | 128 | 100 | 100 | 100 | 0.00 | 56 | 55 | 65 | 18.18% | 6.14% |
| S38584 | 128 | 100 | 100 | 100 | 0.00 | 536 | 549 | 565 | 2.91% | 7.84% |
| S38417 | 128 | 100 | 100 | 100 | 0.00 | 1,133 | 1,124 | 1,115 | -0.80% | 6.75% |
| b17 | 128 | 99.92 | 99.91 | 99.90 | -0.11 | 2,542 | 2,516 | 2,516 | 1.71% | 4.23% |
| b18 | 128 | 99.54 | 99.60 | 99.58 | -0.02 | 5,086 | 5,112 | 5,116 | 0.08% | 1.51% |
| b19 | 128 | 99.65 | 99.65 | 99.64 | -0.01 | 9,395 | 9,387 | 9,398 | 0.12% | 0.80% |

# Analysis

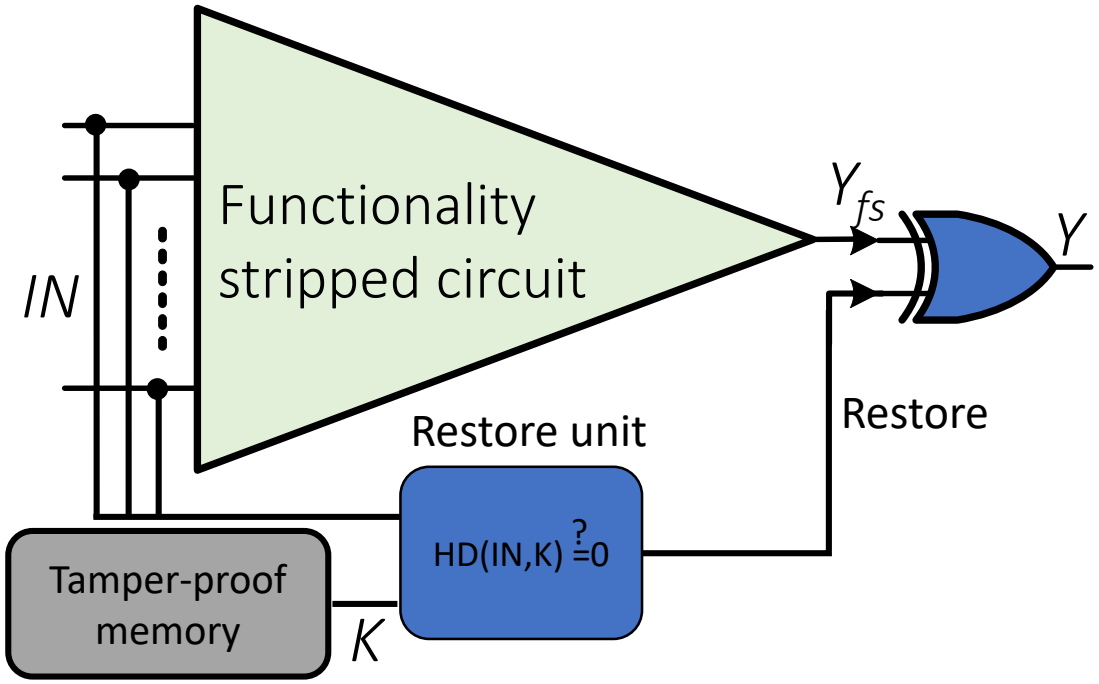
- Attacks
  - SAT-Based Attacks
  - Brute-force Attacks
  - Tampering
- Area Overhead
  - Primarily from the Secure Cell
  - Approximate gate count
    - 5200 for a 256-bit Key
    - 2700 for a 124-bit Key
  - Well below 1%
  - Need one additional pin (*Test*)

# Countermeasure 1- Summary

- We have presented a novel SC design for implementing DFS infrastructure to prevents the leaking of obfuscation key to an adversary, and thus establishes trust in semiconductor manufacturing.
- The proposed infrastructure does not impact the existing IC manufacturing flow.
- The secure cell holds its previous state during manufacturing tests.
  - No leakage of key
- Minimum area overhead
  - Require one additional Test pin

# Countermeasure 2: SFLL-HD$^h$

- SFLL-HD$^h$ structure: striped functionality from the original circuit and then using a flip signal to correct the functionality.

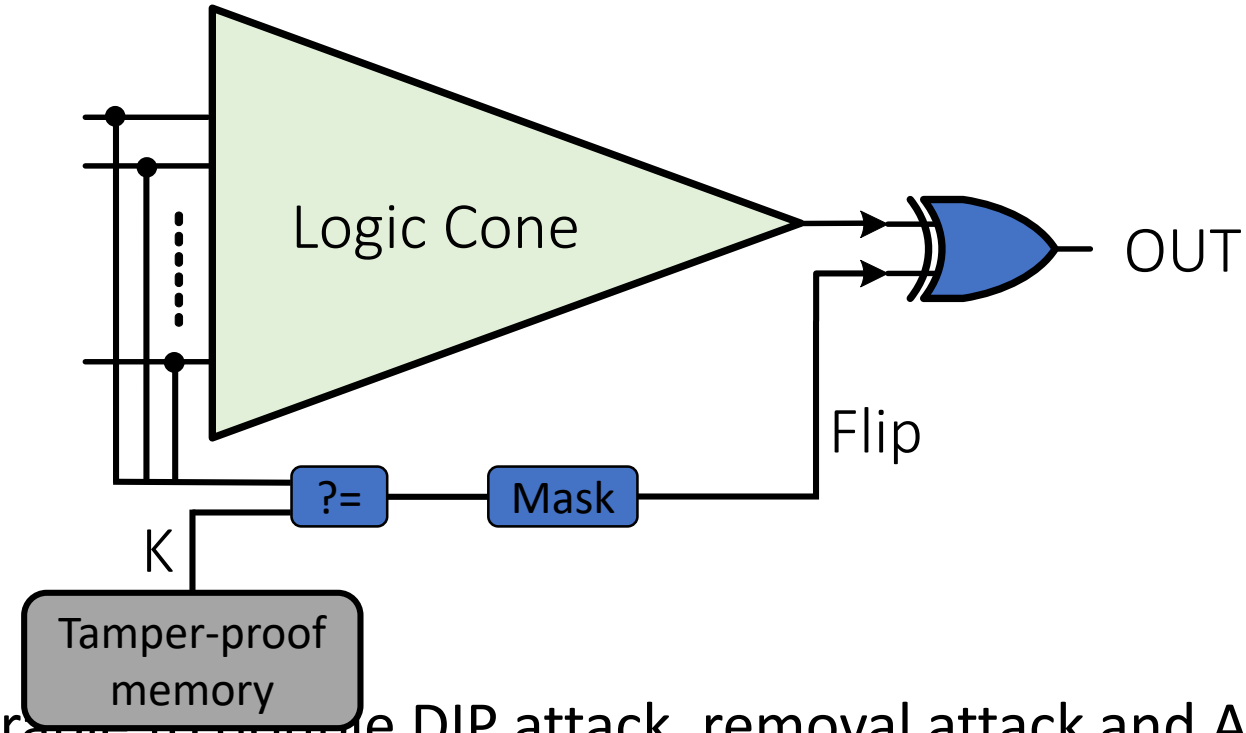



| $Y_{fs}$ | IN | $K_0$ | $K_1$ | $K_2$ | $K_3$ | $K_4$ | $K_5$ | $K_6$ | $K_7$ |
|---|---|---|---|---|---|---|---|---|---|
| √ | 0 | × | √ | √ | √ | √ | √ | √ | √ |
| √ | 1 | √ | × | √ | √ | √ | √ | √ | √ |
| √ | 2 | √ | √ | × | √ | √ | √ | √ | √ |
| √ | 3 | √ | √ | √ | × | √ | √ | √ | √ |
| √ | 4 | √ | √ | √ | √ | × | √ | √ | √ |
| √ | 5 | √ | √ | √ | √ | √ | × | √ | √ |
| × | 6 | × | × | × | × | × | × | √ | × |
| √ | 7 | √ | √ | √ | √ | √ | √ | √ | × |

SFLL-HD$^h$ architecture for h = 0 and k=3

# Countermeasure 3: SARLock

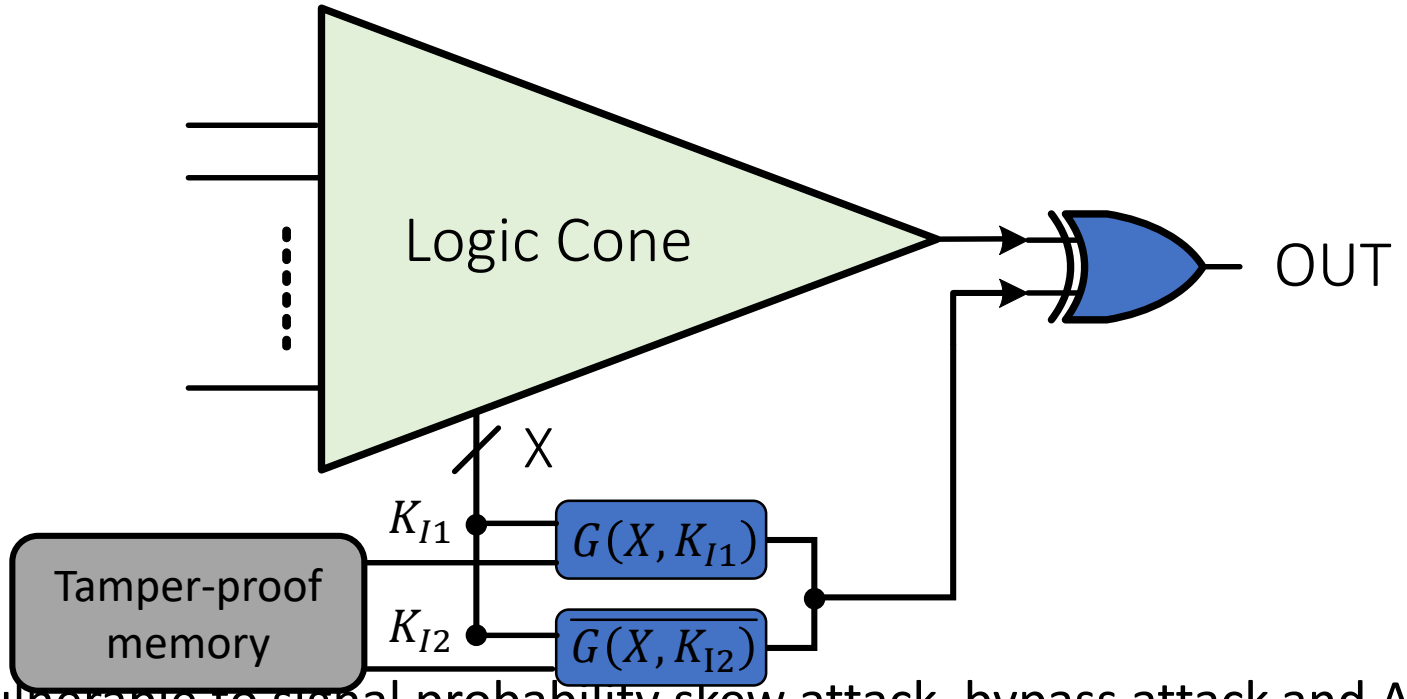
- SARLock structure: add a small comparator circuit in the design to ensure the exponential complexity of key bits.




Cons: Vulnerable to double DIP attack, removal attack and AppSAT attack.

# Countermeasure 4: Anti-SAT

- Anti-SAT structure: uses two key vectors resulting in a total key length of 2n replicating the SARLock truth-table $2^n$ times.




Cons: Vulnerable to signal probability skew attack, bypass attack and AppSAT attack.

# Provable Security for Logic Locking

# Games Cryptographers Play: IND-CPA

Defender

Choose $b \in \{0,1\}$;
Secret key K

Enc($M_b$, K)

Enc($M_b$', K)

...

$M_0$, $M_1$

$M_0$', $M_1$'

...

b = 1

Attacker

Attacker wins if she can guess b with probability better than 0.5

# Proposed Game: IND-LL

Defender

Choose $b \in \{0,1\}$;
Locking key K

Lock($C_b$, K)

Lock($C_b$', K)

...

$C_0$, $C_1$

$C_0$', $C_1$'

...

Attacker

b = 1

Attacker wins if she can guess b with probability better than 0.5

# IND-LL Simulates All Attacks

- All structural and passive functional attacks can be carried out by attacker as she has locked circuits
- All active functional attacks can also be carried out: just run the same algorithm twice with the two locked circuits

**Important to have security proofs that generalize beyond resistance to a specific attack or set of attacks**

# Community Research Challenge

- Develop a locking method that can win the IND-LL game

# In Summary

- Logic locking seems to be taking off
- Current methods not as secure as common crypto primitives
- Opportunity to develop solutions that stand the test of time

# Shameless Plug

- I have open positions for PhD and MS students interested in provable security and formal methods in general and specifically in logic locking
- Send me an email if you are interested: spramod@cse.iitk.ac.in

# Potpourri of Related Theory Issues (1/3)

- Q: Didn’t Barak et al. prove that obfuscation is impossible?
- A: No, they consider a scenario where a program P is obfuscated to generate P’ without the notion of a locking key. We have an additional input: the locking key. So their results are inapplicable. Crypto is full of surprising results and this doesn’t mean obfuscation isn’t impossible, but AFAIK nobody has proved it so.

# Potpourri of Related Theory Issues (2/3)

- Q: Isn't obfuscation the same as functional encryption, which is known to be extremely inefficient?
- A: Not quite, functional encryption is about operating on encrypted data and allows sensitive computation to be performed on untrusted hardware. This is stronger than the locking threat model, which for instance, doesn't allow probing of internal wires while computing on an activated IC.

# Potpourri of Related Theory Issues (2/3)

- Q: Isn't the IND-LL game is too strong to model a passive adversary?
- A: Yes, and this yet another topic for where further research is need.

# SAT Attack Tool Demo

https://bitbucket.org/spramod/host15-logic-encryption