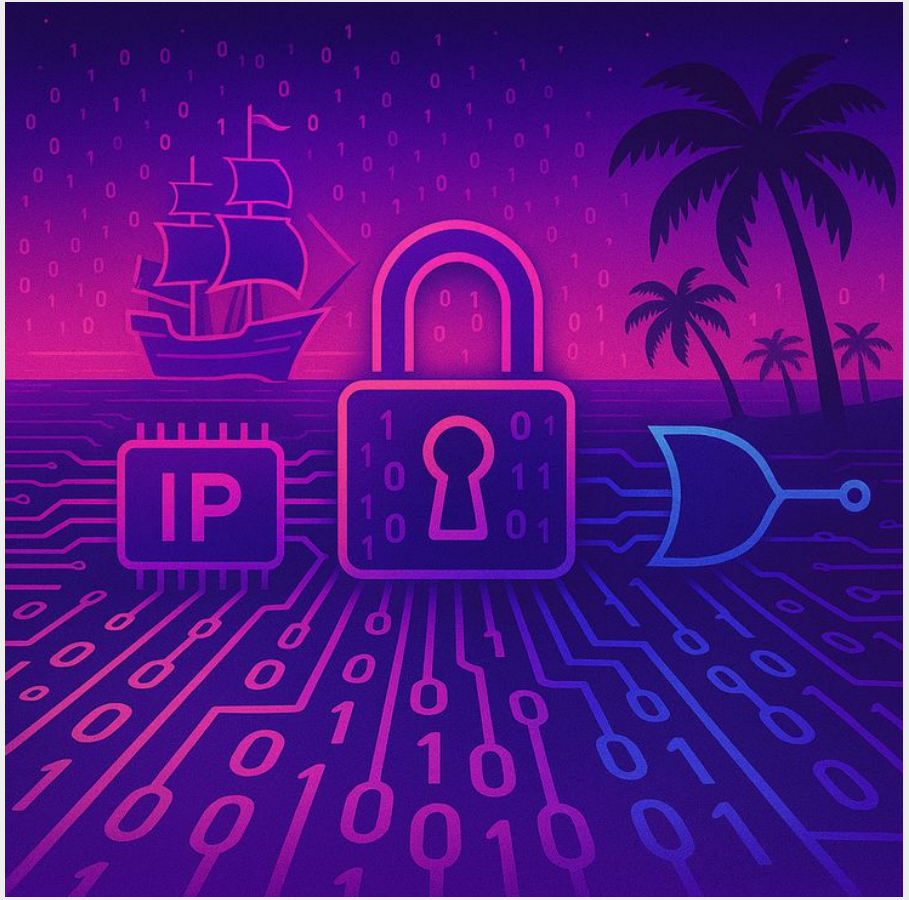
Hardware Security
Week 8 - Piracy and Security
Reverse Engineering, Overbuilding, and Logic Locking



26T1

Hammond Pearce

Slide material acknowledgements to
Benjamin Tan, Ramesh Karri,
JV Rajendran, Jason Blocklove
Christian Pilato, Luca Collini

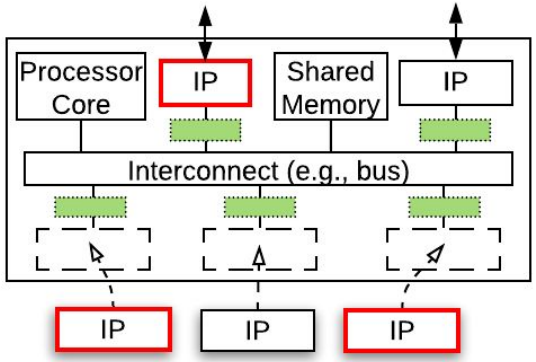
RECAP

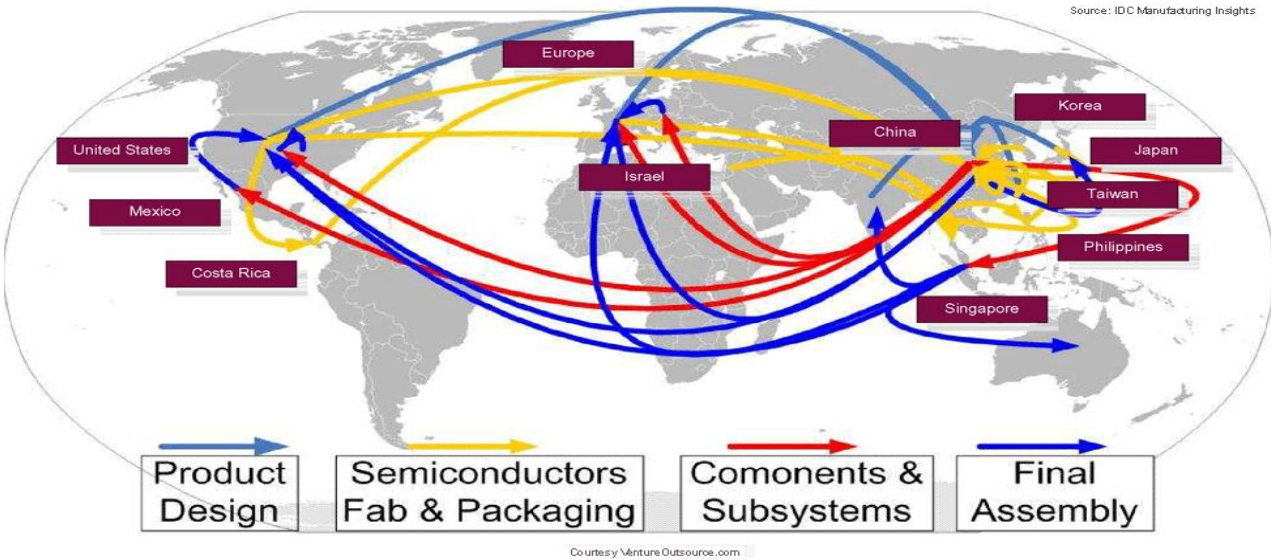
1. What is a root of trust? Why is it important?
2. What is a security objective? What's an example?
3. What's a functional bug vs. a security bug?
4. What's a hardware CWE? CVE? CVSS?
5. How could a hardware Trojan enter an SoC?

Outline

- **Introduction and Motivation**
- Logic Encryption
- Security of Logic Encryption
  - Sensitization Attack
  - SAT Attack
  - Test-data Mining Attack
- Summary

Recap: Globalization of the IC supply chain





- Economic concerns
- Time-to-market
- Design complexity

Security vulnerabilities?

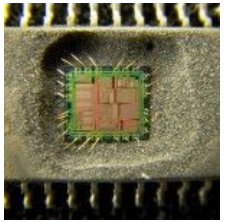
Why protect IPs?

- IPs come in various shapes/forms:
  - Soft IP: RTL description, ready for synthesis
  - Firm IP: gate-level descriptions, ready for hardware
  - Hard IP: synthesized, physical layout---GDSII
- Re-usable, pre-verified
  - Takes a lot of effort to produce $$$ valuable!

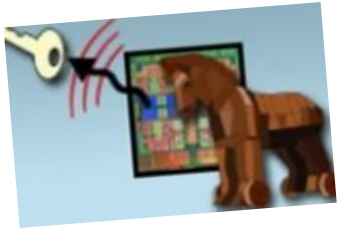
Recap: Security vulnerabilities and trust issues

Counterfeiting

IP Piracy



Reverse Engineering



Hardware Trojans

Attack Incentives

- What does an attacker want?
  - Use a design they're not supposed to?
  - Make more than they're supposed to, sell the design as their own?
  - .....?

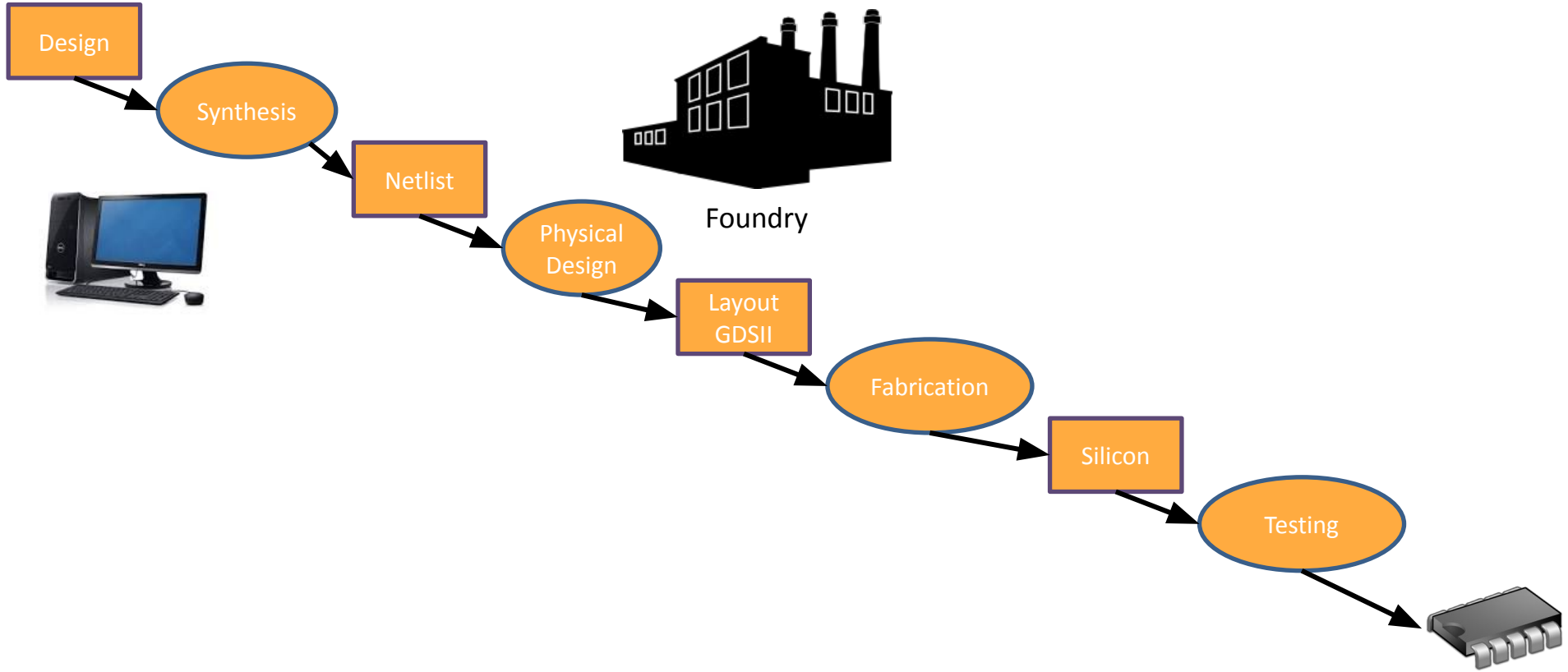
- Can we come up with a way to stop this problem?

Security vulnerabilities and trust issues

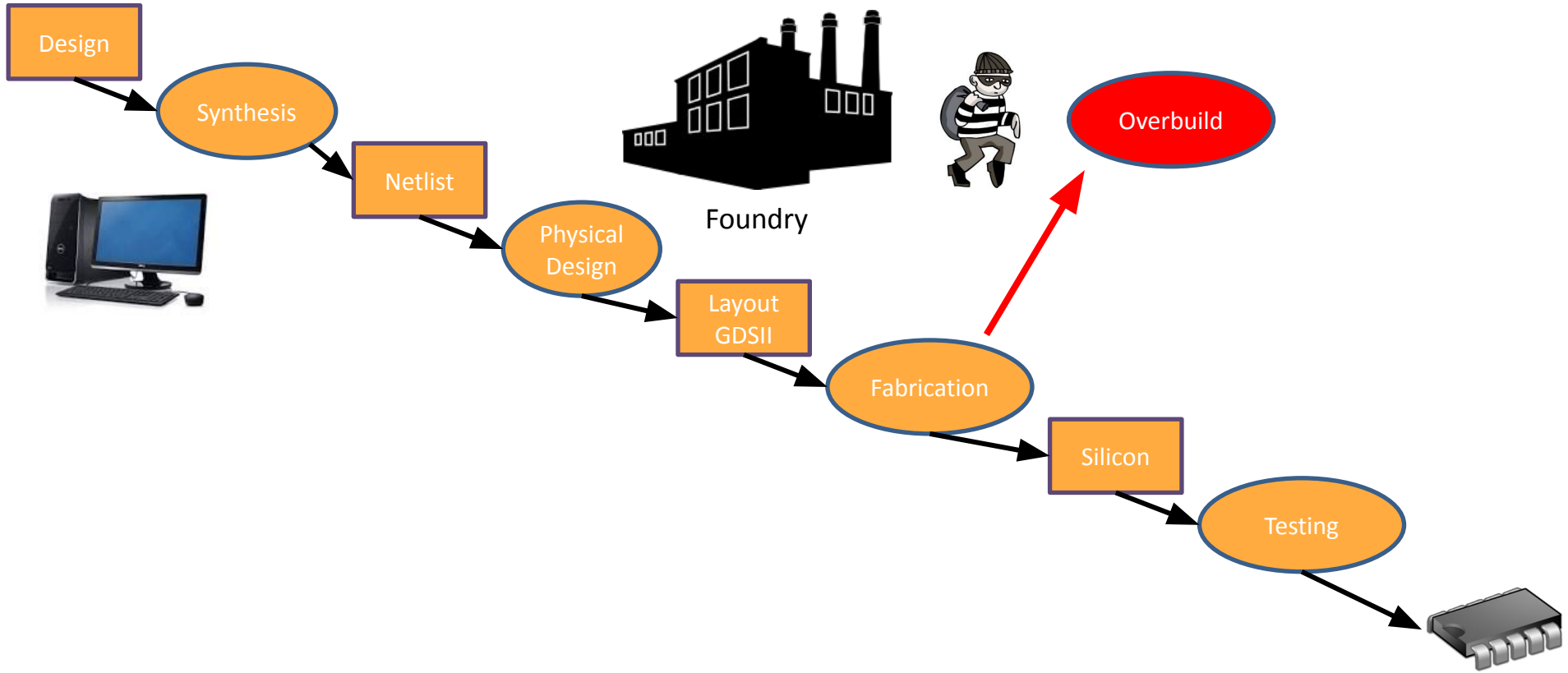
Impact

- Loss of revenue ~$4 billion annually
- Loss of trust
- Unreliable consumer electronics

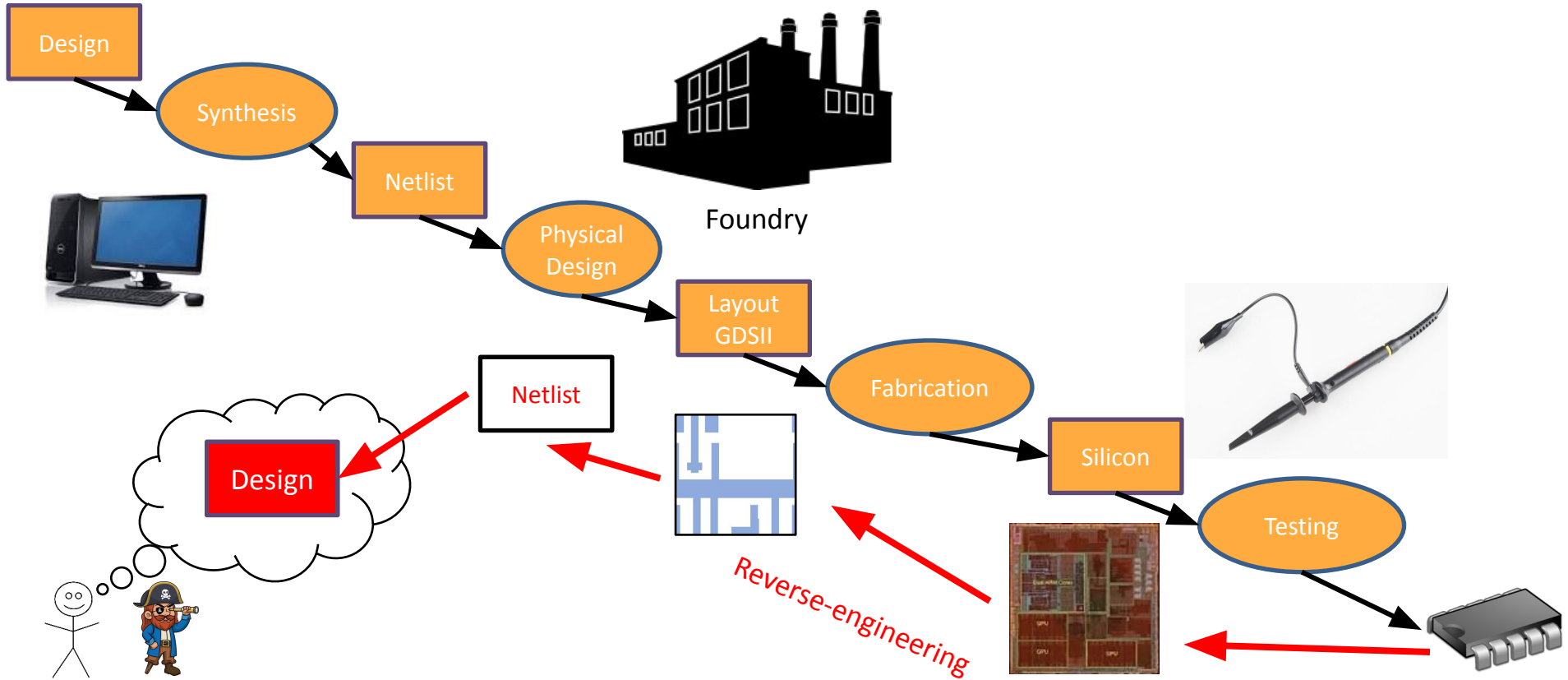
Thinking about threats



Thinking about threats



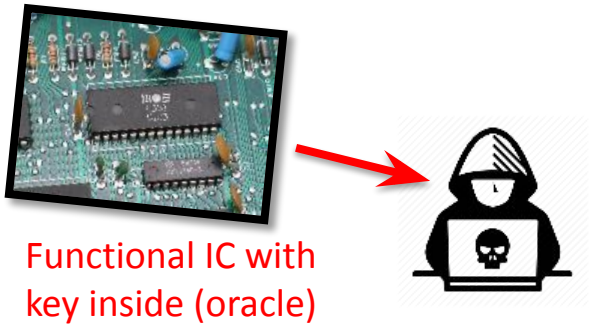
Thinking about threats

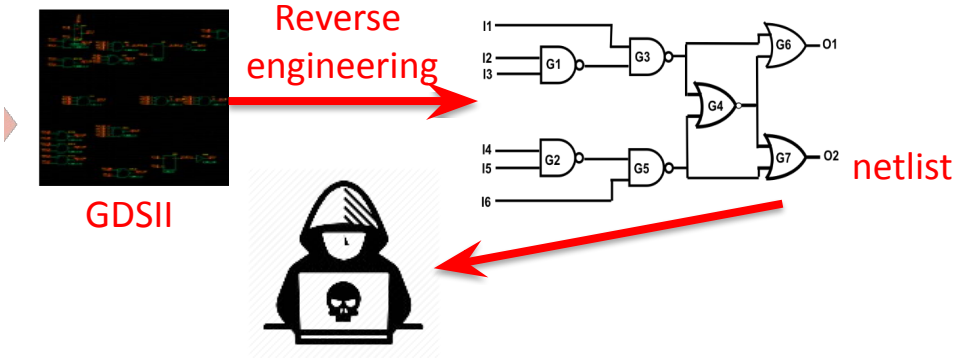


Who could be the "attacker"?

- The "end-user"?
  - Why?
  - What can they do?

- The foundry?
  - Why?
  - What can they do?





Aside: Why do we only consider netlist?

- DfT
- All circuits reduce to:
  - Scan chain
  - Combinational netlist
- Using a DfT scan chain we can extract truth table for circuit
- … this is enough to duplicate the circuit!

Solution - Encryption?

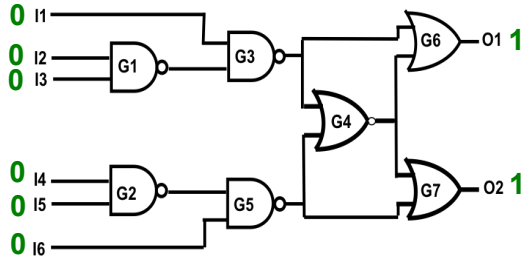
- Can we come up with a way to stop this problem?

- Can we "lock" a circuit with a key?
  - If you have the correct key, you can use the device
  - If you don't have the correct key, it does not work properly

1.J. Roy, et al.,“EPIC: Ending Piracy of Integrated Circuits,” DATE, 2008.

Logic Encryption

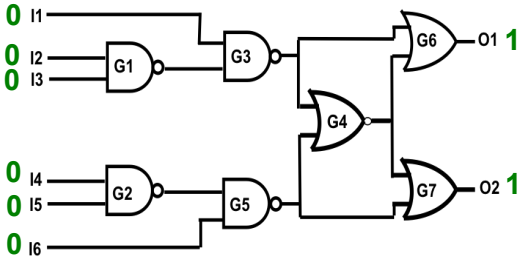
The circuit produces correct output only when the correct key is supplied



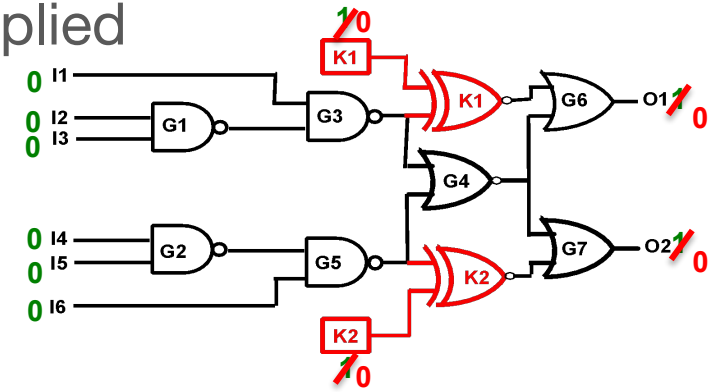
Original netlist

Logic Encryption

The circuit produces correct output only when the correct key is supplied



Original netlist



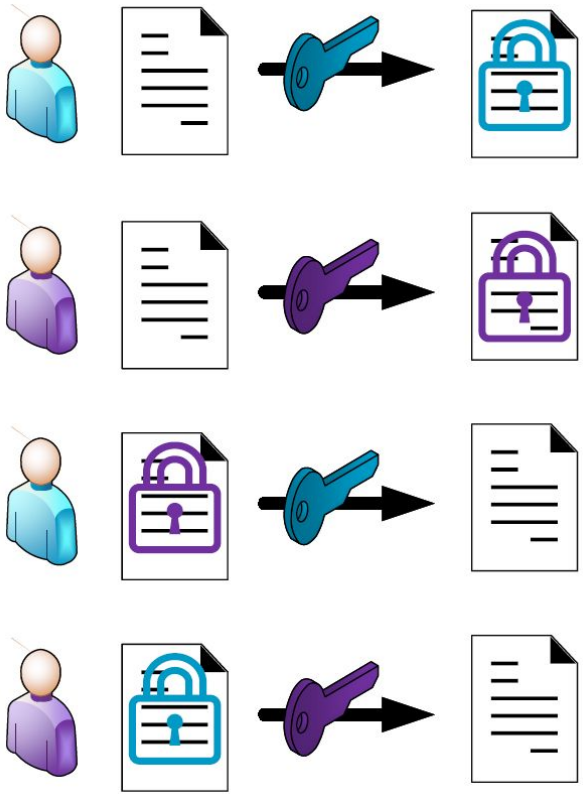
Encrypted netlist

Then what?

- After locking a circuit, how can we distribute the key (for example, to customers)?

- Can we control the number of activated ICs?

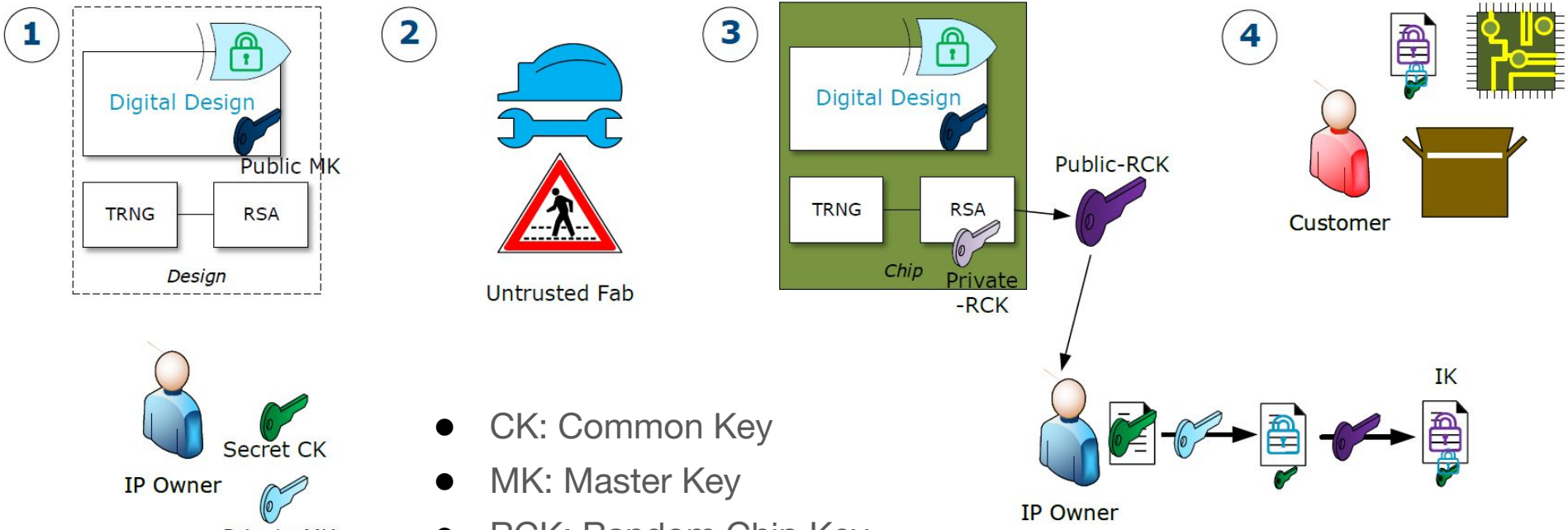
Asymmetric Key Encryption



- Diffie and Hellman

- One key for *encryption* a *different* key for decryption

The EPIC approach



- CK: Common Key
- MK: Master Key
- RCK: Random Chip Key
- IK: Input Key (chip specific)

Scenarios

- Customer tries to use their IK on more chips than they ordered
  - Different chip has different RCKs, so you can't use more than you purchased
- Foundry tries to overbuild
  - Foundry doesn't have the keys, so the IP won't work
- The CK has been stolen
  - A customer can't use it directly because they don’t have access to the private MK to prepare the IK?

What protects the protection?

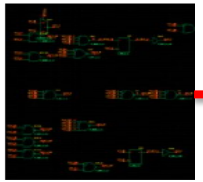
- The CK is an asset! If it's exposed (or broken), the IP can be reverse engineered….
- Where do we put the gates? How do we disguise logic?
- *Problem solved?*

What can attackers do?

Functional IC with key inside (oracle)

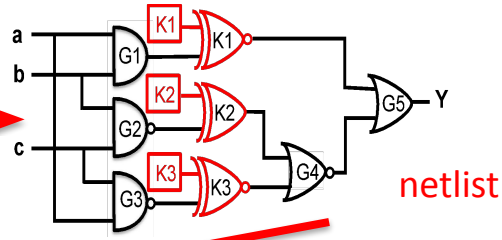
- Apply inputs to the IC
- Collect outputs from the IC
- Infer the key from input-output pairs

brute-force + side-channel analysis



GDSII

Reverse engineering

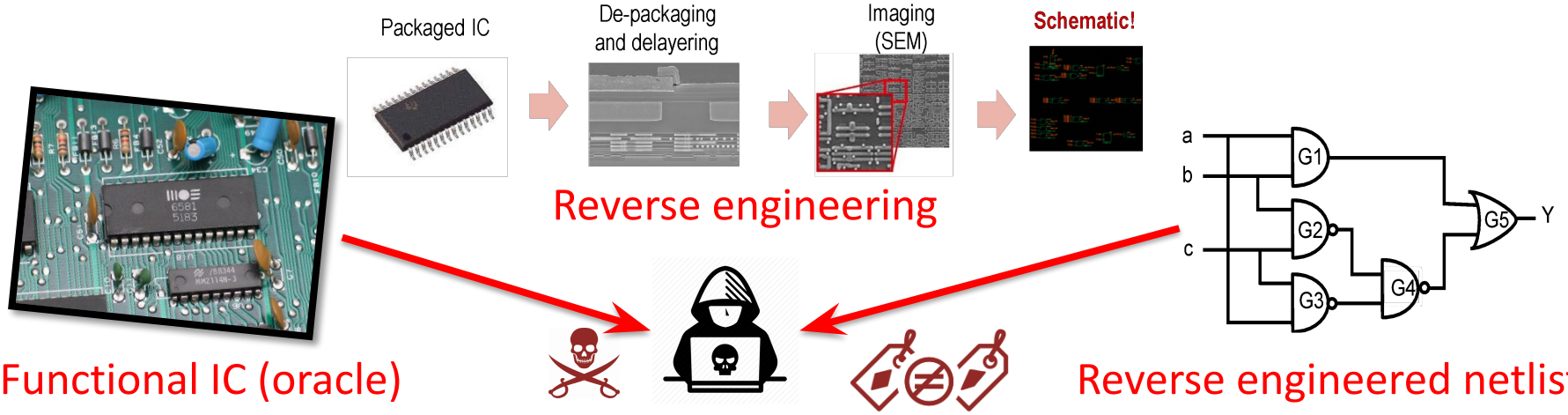


netlist

- Analyze netlist to isolate key logic
- Infer the key from key logic or simply remove key logic to get IP

locked netlist analysis

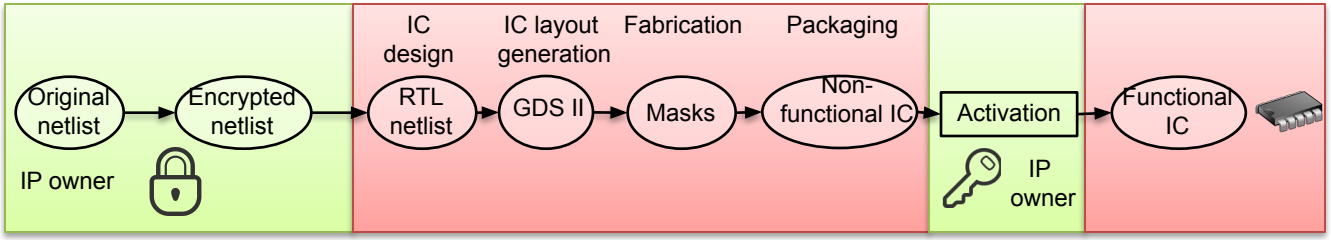
What can attackers do?



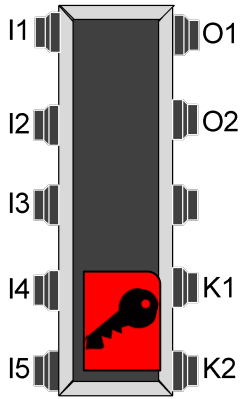
Outline

- Introduction
- **Logic Encryption**
- Security of Logic Encryption
  - Sensitization Attack
  - SAT Attack
  - Test-data Mining Attack
- Summary

Logic encryption (LE)[1]



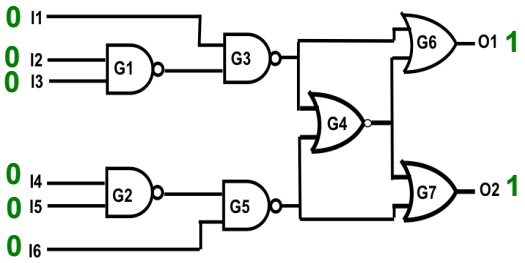
- Design for trust solutions:
  - Watermarking
  - Fingerprinting
  - IC metering
  - Logic encryption
- Logic encryption
  - IP owner encrypts/locks the netlist
  - IC is activated by loading the correct key



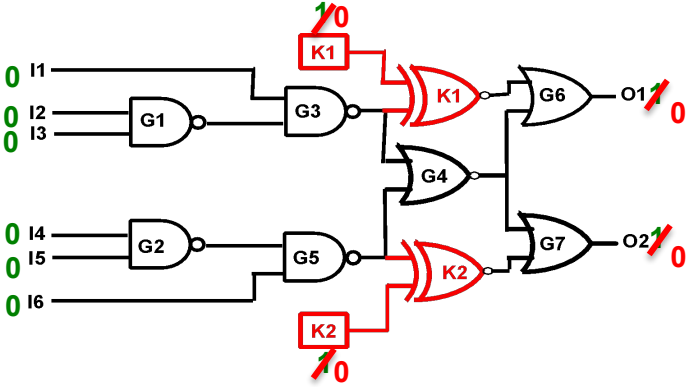
1.J. Roy, et al.,"EPIC: Ending Piracy of Integrated Circuits," DATE, 2008.

Logic Encryption

Circuit produces correct output only when the correct key is supplied



Original netlist



Encrypted netlist

Metrics for logic encryption

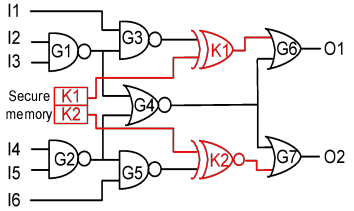
- An attacker should not learn the outputs
  - Hamming distance between outputs of the designs with correct and incorrect keys should be 50%
  - Entropy is maximum at 50%.
  - 0% □ No difference between outputs
  - 100% □ Outputs are complements of each other
- An attacker should not learn the key
  - With polynomial number of input and output pairs

Logic Encryption Techniques

Random LE (RLE)[1]

Key-gates at locations to minimize delay overhead (random from security perspective)

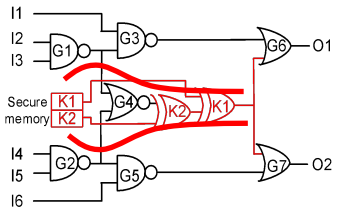
Key-gates uniformly distributed in the netlist



Fault analysis based LE (FLE)[2]

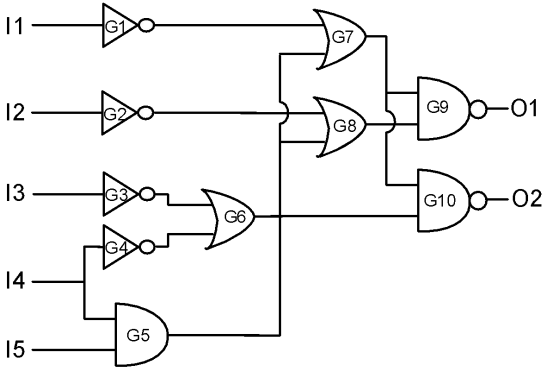
Key-gates at the most influential locations in the netlist

Key-gates tend to be localized and mostly back-to-back

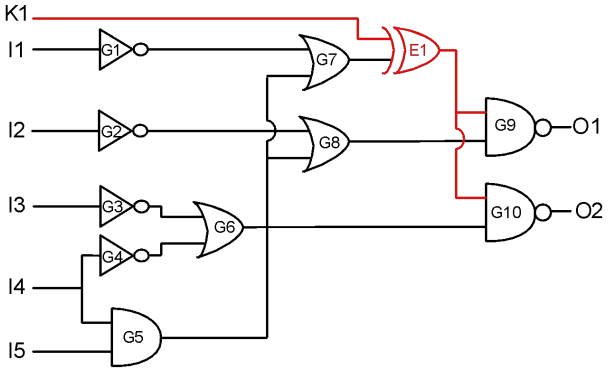


1. J. Roy, et al. ,“EPIC: Ending Piracy of Integrated Circuits,” DATE, 2008.
2. J. Rajendran, et al. ,”Fault-Analysis based Logic Encryption”, TCOMP 2015
3. J. Rajendran, et al. ,"Security Analysis of Logic Obfuscation”, DAC 2012

Working



(a) Original circuit



(b) Circuit encrypted by inserting one XOR gate (E1)

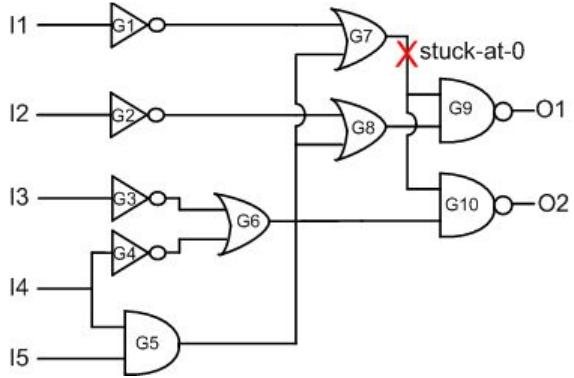
- A correct output is produced for the input pattern ‘01110’ even if the key is invalid (K1 =1)
- The effect of wrong key is blocked by gates G9 and G10

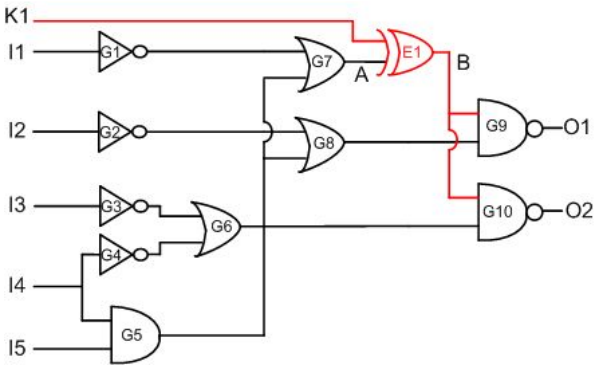
Logic Encryption Vs IC Testing

Logic Encryption	VLSI Testing
Wrong Key	Fault excitation
Effect on output	Fault propagation
Multiple wrong keys	Multiple faults
Self-cancellation of wrong keys	Fault masking

Logic Encryption Vs IC Testing

Logic Encryption	VLSI Testing
Wrong Key	Fault excitation
Effect on output	Fault propagation
Multiple wrong keys	Multiple faults
Self-cancellation of wrong keys	Fault masking

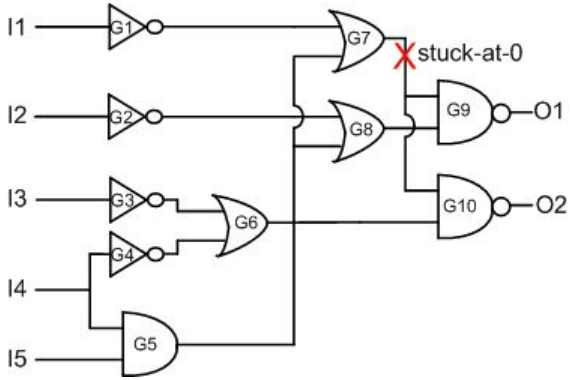


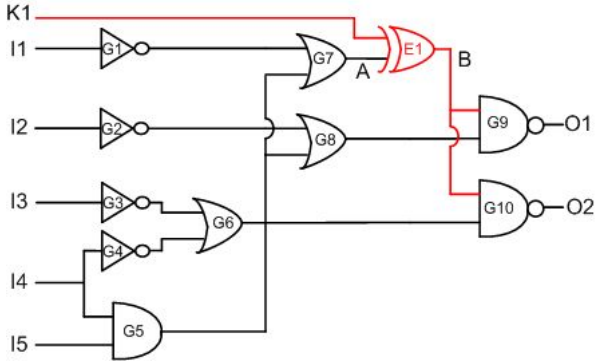


For a wrong key (K1=1) the value of net B is NOT(A). This is same as exciting a s-a-0 (when A=1) or s-a-1 (when A=0) fault at the output of G7.

Logic Encryption Vs IC Testing

Logic Encryption	VLSI Testing
Wrong Key	Fault excitation
Effect on output	Fault propagation
Multiple wrong keys	Multiple faults
Self-cancellation of wrong keys	Fault masking

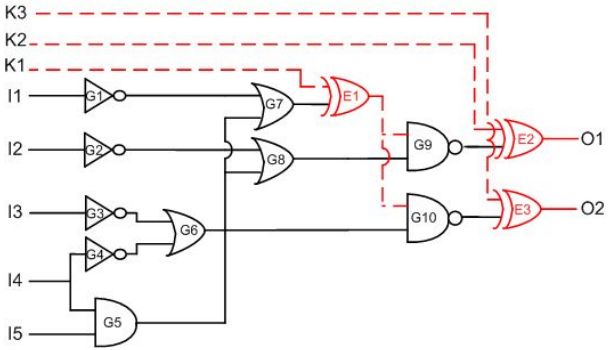


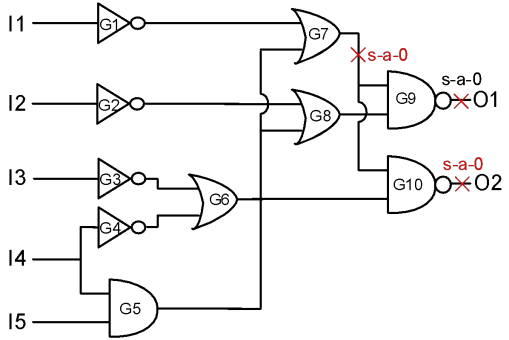


- Effect of s-a-0 is observed at O1 and O2 by justifying the other inputs of G9 and G10
- Effect of wrong key is observed at O2 and O2 by justifying the other inputs of G9 and G10

Logic Encryption Vs IC Testing

Logic Encryption	VLSI Testing
Wrong Key	Fault excitation
Effect on output	Fault propagation
Multiple wrong keys	Multiple faults
Self-cancellation of wrong keys	Fault masking

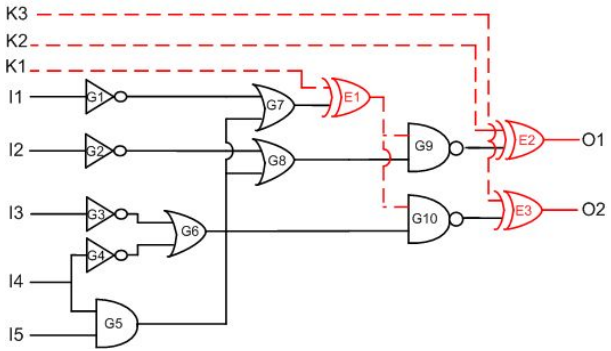


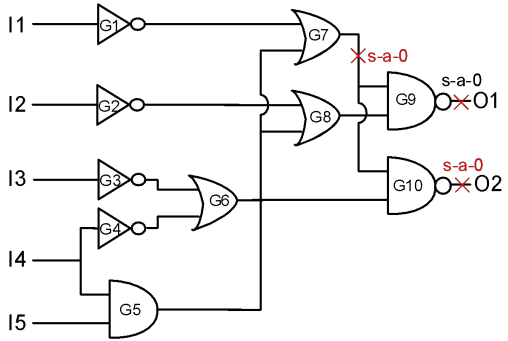


- One key-gate □ either a s-a-0 or s-a-1
- Multiple key-gates □ multiple s-a-0 or s-a-1

Logic Encryption Vs IC Testing

Logic Encryption	VLSI Testing
Wrong Key	Fault excitation
Effect on output	Fault propagation
Multiple wrong keys	Multiple faults
Self-cancellation of wrong keys	Fault masking





- Effect of a fault at G7 can be blocked by G9 and G10
- Effect of wrong key at E1 can be blocked by wrong keys at E2 and E3

Breakout Session

Instructions

1. Ask each other how you're doing, catch up
2. Create a circuit made of 6 logic gates
3. Work out how many places one could put key gates
4. Randomly pick places to insert 2 key gates, work out the output corruption
5. Think about how many more gates you need to add to reach 50% corruption
6. Think about how to decide how many gates to add, what sort of trade offs

When we're done, you will report back on points 2-6

Discourse: Look at an answer or two

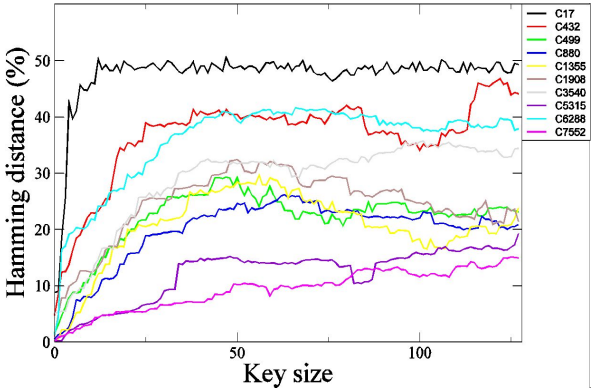
Fault-impact based logic encryption

- Where to insert gates?
  - Insert gates at places where, if faults occur, they can be excited and show at the output
- How many to insert?
  - Until 50% of the outputs are corrupted for any random key

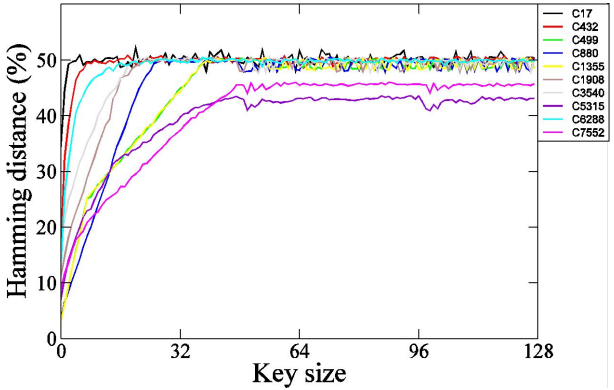
$$\text{Fault Impact} = (\#\text{Test Patterns}_{\text{s-a-0}} \times \#\text{O/Ps}_{\text{s-a-0}}) + (\#\text{Test Patterns}_{\text{s-a-1}} \times \#\text{O/Ps}_{\text{s-a-1}})$$

Results

Hamming distance b/w outputs of designs with correct key and a random incorrect key



(a) EPIC– Delay-based insertion



(b) Fault-analysis based encryption

Fault-analysis based insertion achieves 50% Hamming distance for ISCAS 85 benchmarks

Attacks on Logic Encryption

How Secure is Logic Encryption?

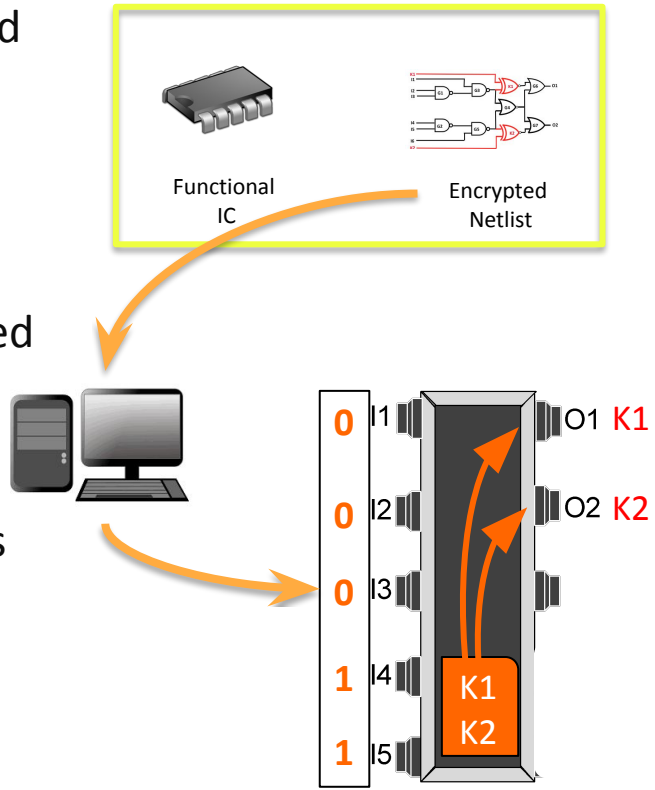
Goal: Determine the secret key used for logic encryption

Attacker has:

- Encrypted netlist
- Functional IC (with embedded key)

Attacker does:

- Compute the attack patterns from the encrypted netlist
- Applies them on IC
- Infers key from responses



J. Rajendran, et al. "Security analysis of logic obfuscation", DAC 2012

Outline

Sensitization attack	SAT attack	Test-data mining attack
Threat Model Encrypted netlist Functional IC	**Threat Model** Encrypted netlist Functional IC	**Threat Model** Encrypted netlist Test data
Attack method Sensitize individual key bits to primary outputs	**Attack method** Eliminate incorrect keys using “distinguishing input patterns”	**Attack method** Use test data to extract the secret key for pre-test activation
Defense Strong Logic Encryption	**Defense** SARLock	**Defense** Post-test activation

1. Rajendran, Jeyavijayan et al. "Security analysis of logic obfuscation”, DAC 2012
2. Yasin et al. “Activation of Logic Encrypted Chips: Pre-Test or Post-Test?.”, DATE 2016
3. Subramanyan et al. “Evaluating the Security of Logic Encryption Algorithms”, HOST 2015

Outline

Sensitization attack

Threat Model
Encrypted netlist
Functional IC

Attack method
Sensitize individual key bits to primary outputs

Defense
Strong Logic Encryption

SAT attack

Threat Model
Encrypted netlist
Functional IC

Attack method
Eliminate incorrect keys using “distinguishing input patterns”

Defense
SARLock

Test-data mining attack

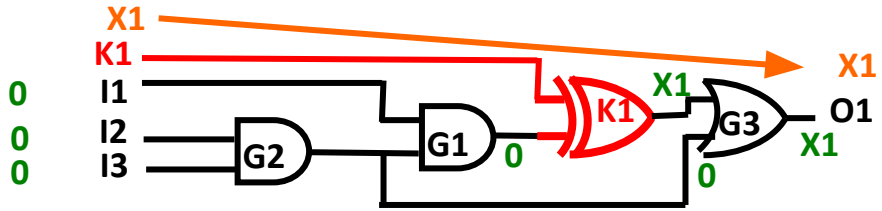
Threat Model
Encrypted netlist
Test data

Attack method
Use test data to extract the secret key for pre-test activation

Defense
Post-test activation

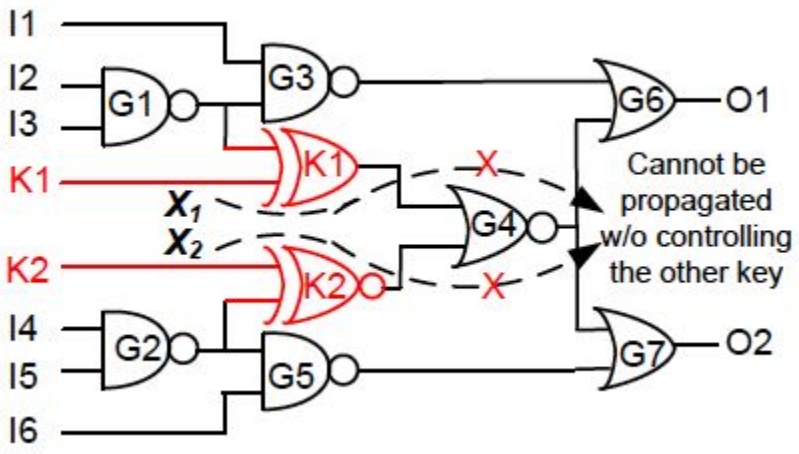
1. Rajendran, Jeyavijayan et al. "Security analysis of logic obfuscation”, DAC 2012
2. Yasin et al. “Activation of Logic Encrypted Chips: Pre-Test or Post-Test?.”, DATE 2016
3. Subramanyan et al. “Evaluating the Security of Logic Encryption Algorithms”, HOST 2015

Attack 1: Sensitization Attack



- **Objective**: Sensitize key K1 to primary output O1
- Find a test pattern to do sensitization
- Apply the test pattern to functional IC and observe the responses to find the value of key

Solution: Strong Logic Encryption (SLE)



- Individual sensitization is not possible
- Pairwise secure key-gates
- Requires brute force:
  - Enumerate all possible values for the key bits
  - Exponential complexity!

Outline

- Introduction and Motivation
- Logic Encryption
- Security of Logic Encryption
  - Sensitization Attack
  - **SAT Attack**

Outline

Sensitization attack

Threat Model
Encrypted netlist
Functional IC

Attack method
Sensitize individual key bits to primary outputs

Defense
Strong Logic Encryption

SAT attack

Threat Model
Encrypted netlist
Functional IC

Attack method
Eliminate incorrect keys using “distinguishing input patterns”

Defense
SARLock

Test-data mining attack

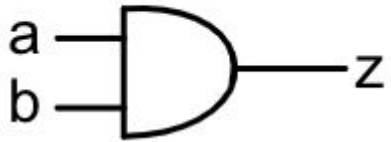
Threat Model
Encrypted netlist
Test data

Attack method
Use test data to extract the secret key for pre-test activation

Defense
Post-test activation

SAT attack: Overview

- SAT = Boolean SATisfiability
  - Find an assignment to variables that makes a Boolean formula evaluate to true.
  - Formulas represented in product-of-sum form aka conjunctive normal form (CNF).
  - Converting to CNF - Tseytin transformation

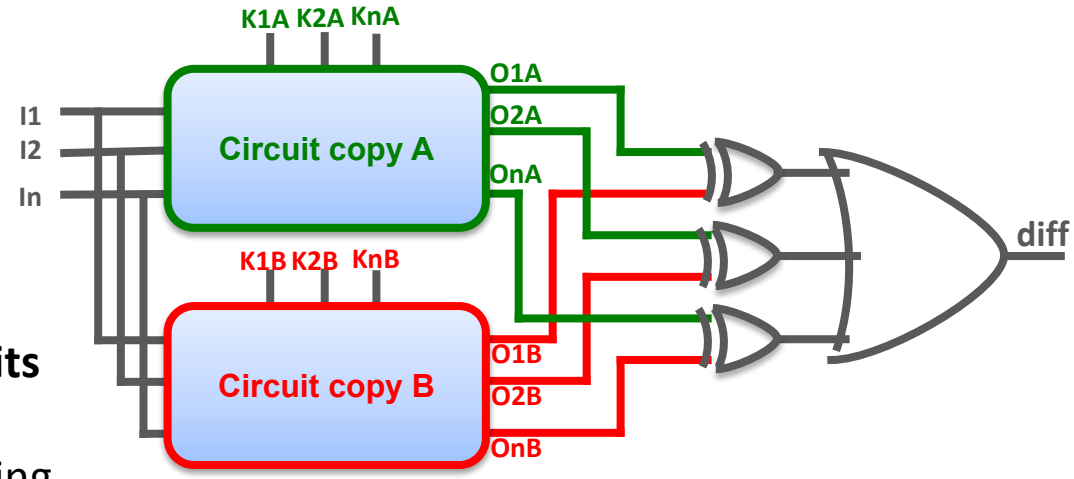


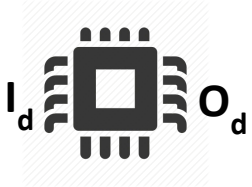
$$
\begin{aligned}
z &\iff a \,\&\, b \\
&= (z \implies a \,\&\, b) \,\&\, (a \,\&\, b \implies z) \\
&= (!z \mid a \,\&\, b) \,\&\, (!(a \,\&\, b) \mid z) \\
&= (!z \mid a) \,\&\, (!z \mid b) \,\&\, (z \mid !a \mid !b) \\
&= (!z \mid a)\ (!z \mid b)\ (z \mid !a \mid !b)
\end{aligned}
$$

  - SAT returns assignment A = 1, B = 1.

Distinguishing input pattern

- A miter circuit is created
  - Two copies of the circuit XORed
  - Inputs are shorted
  - Keys are left separate
  - Convert to CNF
  - Fed to the SAT solver
  - **Find an input such that two circuits produce different outputs**
  - Such an input is called distinguishing input pattern (DIP)
  - diff =1 iff **OA(I_d,KA) ≠ OB(I_d,KB)**
  - At most one output can match O_d
  - At least one key value is incorrect

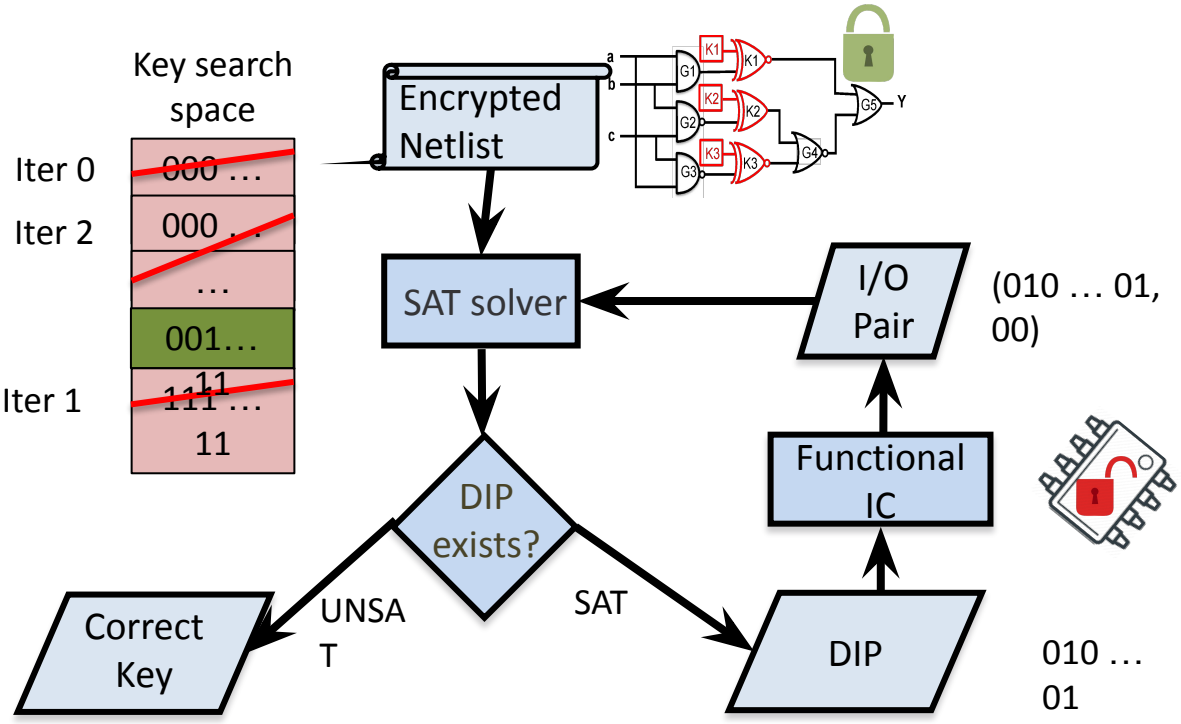




OA == O_d ?

OB == O_d ?

SAT Attack: Overview



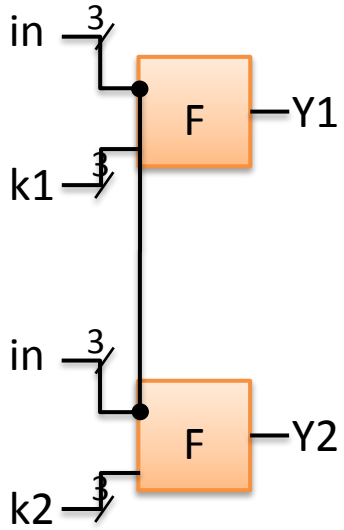
Subramanyan, Ramakrishna. "Evaluating the Security of Logic Encryption Algorithms." HOST 2015

A toy example of SAT attack

Try to see if it's possible to set **in, k1, k2,** such that Y1 != Y2

					Output Y for different key values								
No.	**a**	**b**	**c**	**Y**	**k0**	**k1**	**k2**	**k3**	**k4**	**k5**	**k6**	**k7**	**Pruned key values**
0	0	0	0	0	1	1	1	1	1	1	0	1	
1	0	0	1	0	1	1	1	1	1	1	0	1	
2	0	1	0	0	1	1	1	1	1	1	0	1	
DIP 1	0	1	1	1	1	1	1	1	0	1	1	1	Iter 1: {k4}
DIP 3	1	0	0	0	1	1	1	1	1	1	0	1	Iter 3: all incorrect
5	1	0	1	1	1	1	1	1	1	1	1	0	
6	1	1	0	1	1	1	0	1	1	1	1	1	
DIP 2	1	1	1	1	1	0	0	1	1	1	1	1	Iter 2: {k1, k2}

Attack success ≈ DIP distinguishing ability

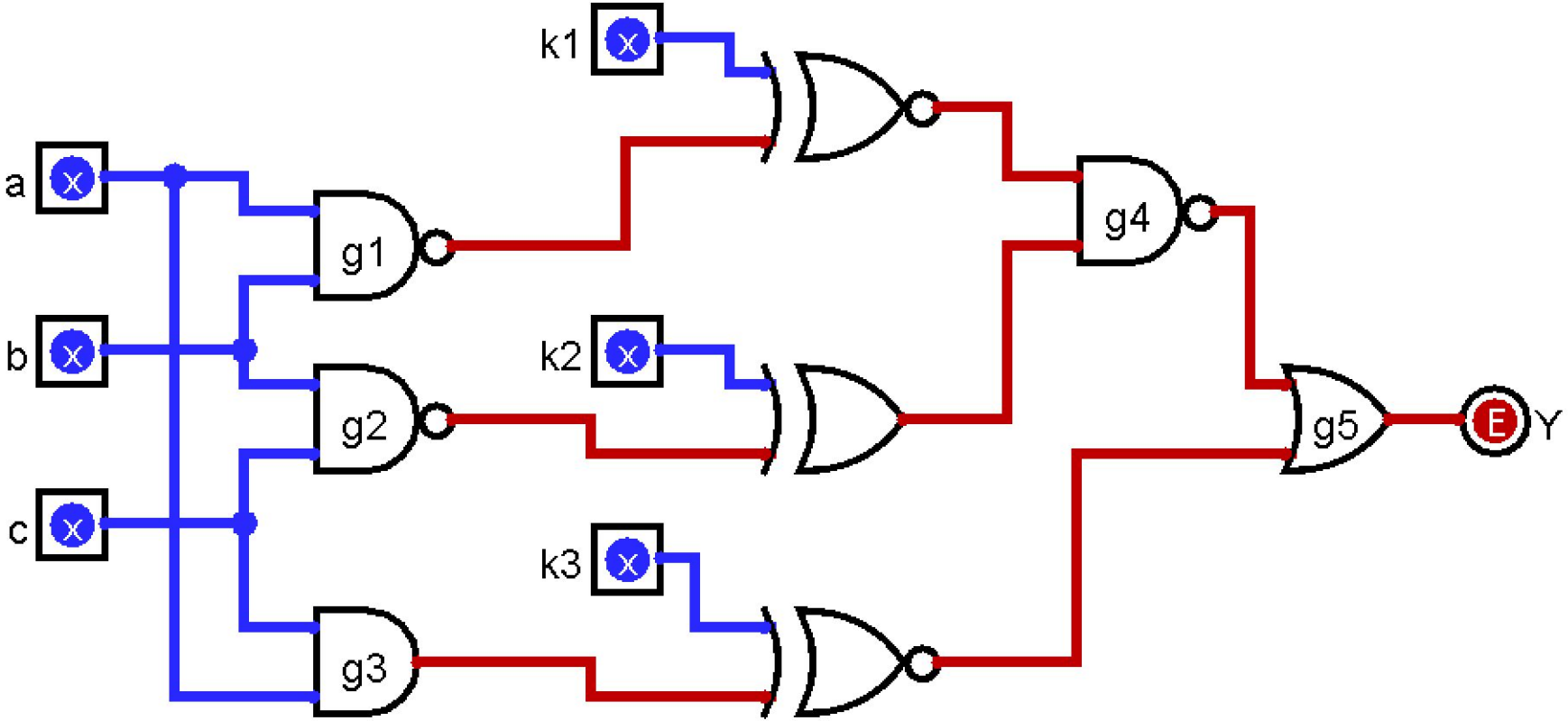


SAT Attack Algorithm

Algorithm 1 Logic Decryption Algorithm

Function: *decrypt*.
Inputs: C and $eval$.
Output: $\vec{\mathtt{K}_\mathtt{C}}$.

1: $i := 1$
2: $F_1 = C(\vec{X}, \vec{K}_1, \vec{Y}_1) \wedge C(\vec{X}, \vec{K}_2, \vec{Y}_2)$
3: **while** $sat[F_i \wedge (\vec{Y}_1 \neq \vec{Y}_2)]$ **do**
4: $\quad \vec{\mathtt{X}}_i^d := sat\_assignment_{\vec{X}}[F_i \wedge (\vec{Y}_1 \neq \vec{Y}_2)]$
5: $\quad \vec{\mathtt{Y}}_i^d := eval(\vec{\mathtt{X}}_i^d)$
6: $\quad F_{i+1} := F_i \wedge C(\vec{\mathtt{X}}_i^d, \vec{K}_1, \vec{\mathtt{Y}}_i^d) \wedge C(\vec{\mathtt{X}}_i^d, \vec{K}_2, \vec{\mathtt{Y}}_i^d)$
7: $\quad i := i + 1$
8: **end while**
9: $\vec{\mathtt{K}_\mathtt{C}} := sat\_assignment_{\vec{K}_1}(F_i)$

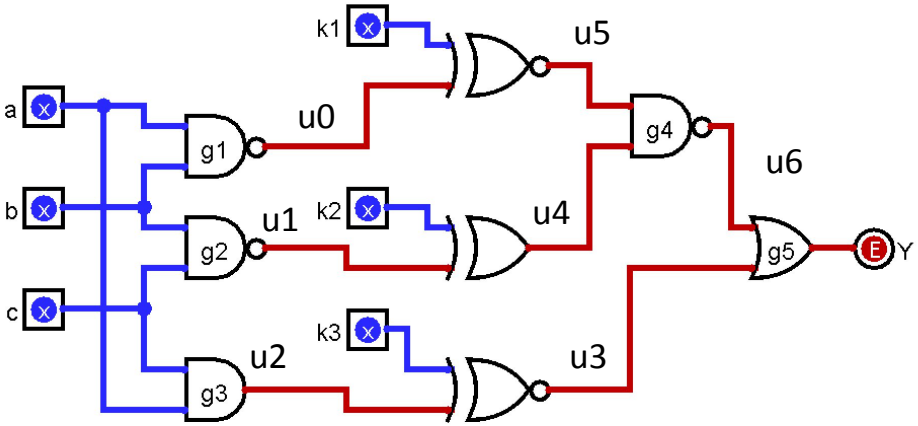

k1
a
g1
g4
b
k2
g2
g5
Y
c
k3
g3

Example

Algorithm 1 Logic Decryption Algorithm

Function: *decrypt*.
Inputs: C and *eval*.
Output: $\vec{\mathrm{K_C}}$.

1: $i := 1$
2: $F_1 = C(\vec{X}, \vec{K}_1, \vec{Y}_1) \wedge C(\vec{X}, \vec{K}_2, \vec{Y}_2)$
3: **while** $sat[F_i \wedge (\vec{Y}_1 \neq \vec{Y}_2)]$ **do**
4: $\quad \vec{\mathrm{X}}_i^d := sat\_assignment_{\vec{X}}[F_i \wedge (\vec{Y}_1 \neq \vec{Y}_2)]$
5: $\quad \vec{\mathrm{Y}}_i^d := eval(\vec{\mathrm{X}}_i^d)$
6: $\quad F_{i+1} := F_i \wedge C(\vec{\mathrm{X}}_i^d, \vec{K}_1, \vec{\mathrm{Y}}_i^d) \wedge C(\vec{\mathrm{X}}_i^d, \vec{K}_2, \vec{\mathrm{Y}}_i^d)$
7: $\quad i := i + 1$
8: **end while**
9: $\vec{\mathrm{K_C}} := sat\_assignment_{\vec{K}_1}(F_i)$



Demo

- http://logictools.org/prop.html

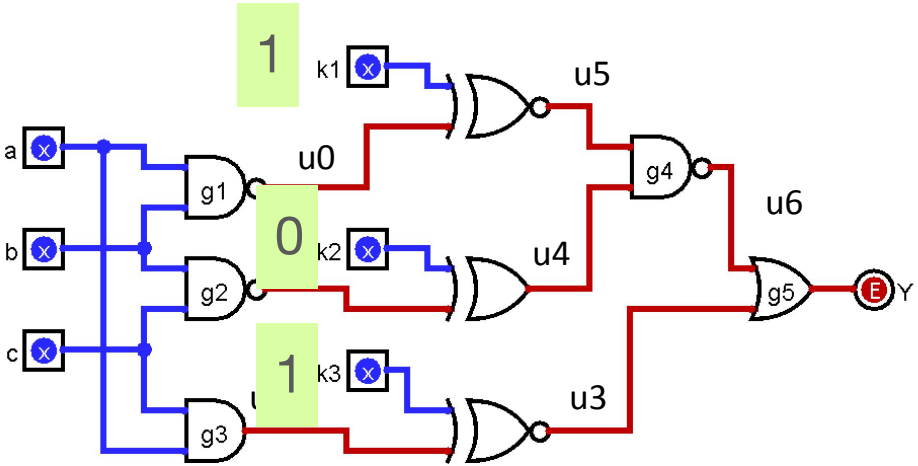
a	b	c	Y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Example

Algorithm 1 Logic Decryption Algorithm

Function: *decrypt*.
Inputs: C and *eval*.
Output: $\vec{\mathrm{K_C}}$.

1: $i := 1$
2: $F_1 = C(\vec{X}, \vec{K}_1, \vec{Y}_1) \wedge C(\vec{X}, \vec{K}_2, \vec{Y}_2)$
3: **while** $sat[F_i \wedge (\vec{Y}_1 \neq \vec{Y}_2)]$ **do**
4: $\quad \vec{\mathrm{X}}_i^d := sat\_assignment_{\vec{X}}[F_i \wedge (\vec{Y}_1 \neq \vec{Y}_2)]$
5: $\quad \vec{\mathrm{Y}}_i^d := eval(\vec{\mathrm{X}}_i^d)$
6: $\quad F_{i+1} := F_i \wedge C(\vec{\mathrm{X}}_i^d, \vec{K}_1, \vec{\mathrm{Y}}_i^d) \wedge C(\vec{\mathrm{X}}_i^d, \vec{K}_2, \vec{\mathrm{Y}}_i^d)$
7: $\quad i := i + 1$
8: **end while**
9: $\vec{\mathrm{K_C}} := sat\_assignment_{\vec{K}_1}(F_i)$



5 min break - Discussion Questions

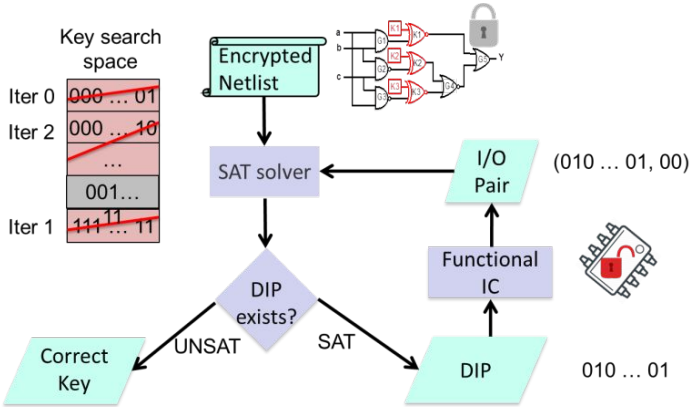
- What is a distinguishing input pattern? What does this pattern distinguish?

- Is it possible that a SAT solver produces two incorrect keys in an iteration of the logic decryption algorithm? Explain why this might/might not occur, and whether this affects the logic decryption algorithm

- Why is the $\overrightarrow{Y_1} \neq \overrightarrow{Y_2}$ clause necessary?

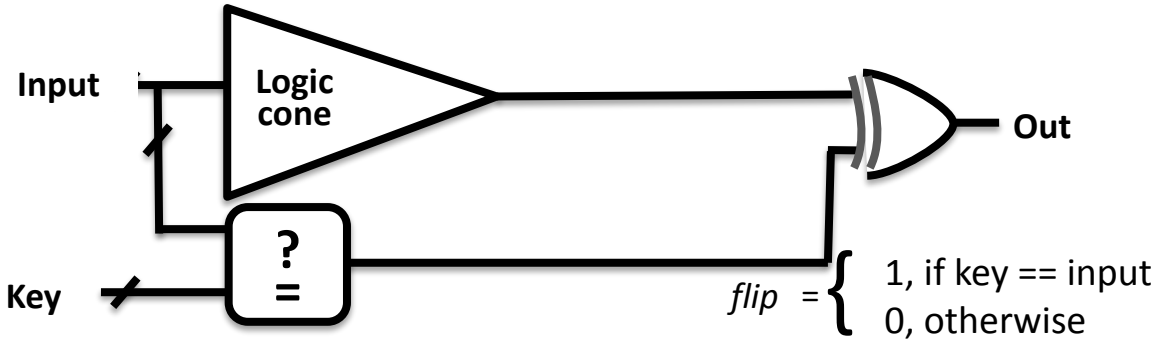
Resisting SAT Attacks – Key Idea

					Output Y for different key values							
No.	a	b	c	Y	k0	k1	k2	k3	k4	k5	k6	k7
0	0	0	0	0	0	1	0	0	0	0	0	0
1	0	0	1	0	1	0	0	0	0	0	0	0
2	0	1	0	0	0	0	0	1	0	0	0	0
3	0	1	1	1	1	1	0	1	1	1	1	1
4	1	0	0	0	0	0	0	0	0	1	0	0
5	1	0	1	1	1	1	1	1	0	1	1	1
6	1	1	0	1	1	1	1	1	1	1	1	1
7	1	1	1	1	1	1	1	1	1	1	1	0



- Desired: Each DIP eliminates one key value
- Number of DIPs = Number of input combinations

SARLock: SAT-attack Resistant LE

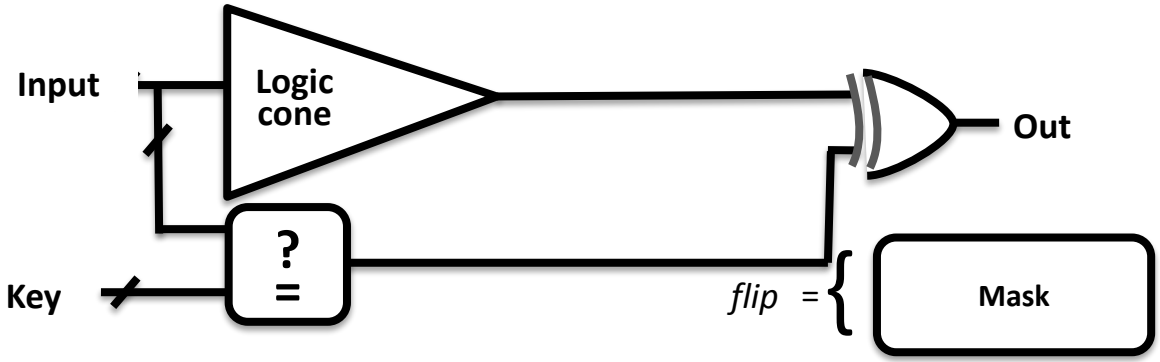


- Flip is asserted for
  - Correct key value
  - Diagonal entries of table

Number of DIPs = 2^K -1

					Output Y for different key values							
No.	a	b	c	Y	k0	k1	k2	k3	k4	k5	k6	k7
0	0	0	0	0	1	0	0	0	0	0	0	0
1	0	0	1	0	0	1	0	0	0	0	0	0
2	0	1	0	0	0	0	1	0	0	0	0	0
3	0	1	1	1	1	1	1	0	1	1	1	1
4	1	0	0	0	0	0	0	0	1	0	0	0
5	1	0	1	1	1	1	1	1	1	0	1	1
6	1	1	0	1	1	1	1	1	1	1	0	1
7	1	1	1	1	1	1	1	1	1	1	1	0

SARLock: SAT-attack Resistant LE



- Mask logic
  - Can be incorporated into logic cone
  - Ensures correct circuit operation

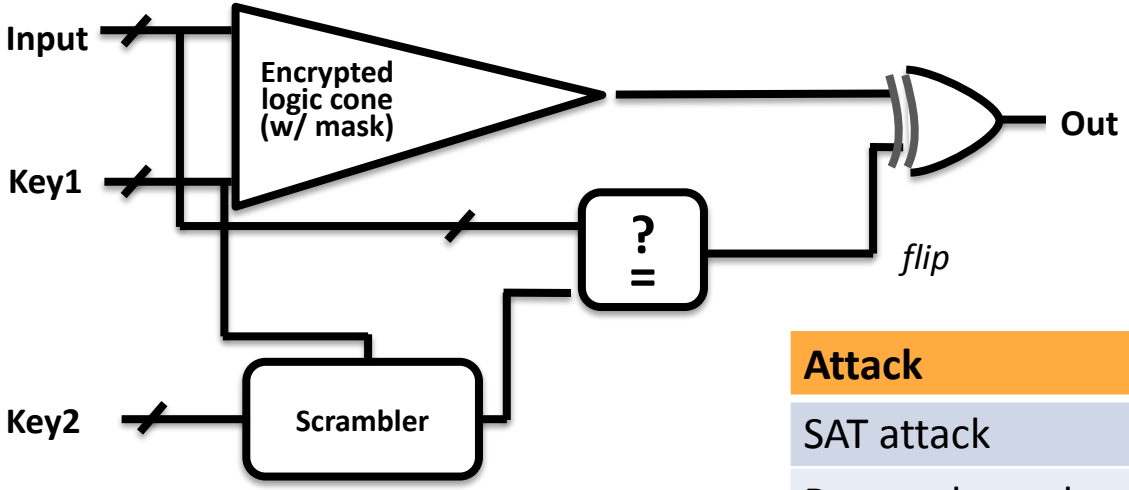
Is SARLock alone Sufficient?

Feature	SARLock
SAT attack resistant?	✓
Removal attack resistant?	✗
Sensitization attack resistant?	?

Integration with SLE to:

- Leverage the strengths of SLE /other logic encryption
- Thwart sensitization and removal attacks

Two-layer Logic Encryption: SARLock + SLE



Attack	Defense
SAT attack	SARLock
Removal attack	SLE
Both	SARLock+SLE

- Scrambler
  - Creates key dependencies to thwart removal attack
  - E.g., Hash functions
- Assumption: Gates used in logic cone, scrambler, comparison, and the final XOR are “indistinguishable”

Experimental Results

- SARLock+SLE
  - Number of DIPs = $2^{|K|}$
  - SARLock **exponentially** increases number of DIPs

Design	#DIPs				Execution Time (s)			
	10	11	12	13	10	11	12	13
s5378	1024	2048	4096	8191	54.1	190.6	619.7	4351.8
c5315	1024	2049	4096	8191	75.4	252.9	829.1	4778.2
c7552	1025	2049	4096	8191	78.3	234.1	757	3165.3
s9234	1027	2049	4102	8195	77.2	247.9	864.1	3225.7
IFU	1023	2056	4100	8206	55.2	166.7	789.5	2309.8
LSUrw	1025	2049	4096	8194	58.2	152	626.9	1802.6
FPUin	1025	2049	4097	8194	28.4	135	1359.6	4497.6
LSUex	1024	2049	4096	8194	52.8	268.3	1137.2	3101.3

SARLock+SLE resists SAT attack

Overhead and Comparison

- SLE+SARLock vs. other techniques
  - Number of key gates: |K| = 64
  - Number of DIPs and execution time are extrapolated

Metrics	RLE	SLE	SarLock+SLE
#DIPs	19	26	4.3E09
Attack time (s)	0.4	0.7	3.1E09
Area overhead (%)	29.6	32.2	35.2
Power overhead (%)	45.0	59.2	61.0
Delay overhead (%)	15.1	17.2	9.3

Attack effort increases exponentially SARLock+SLE

Roy, Jarrod A. et al.,“EPIC: Ending Piracy of Integrated Circuits,” DATE, 2008.

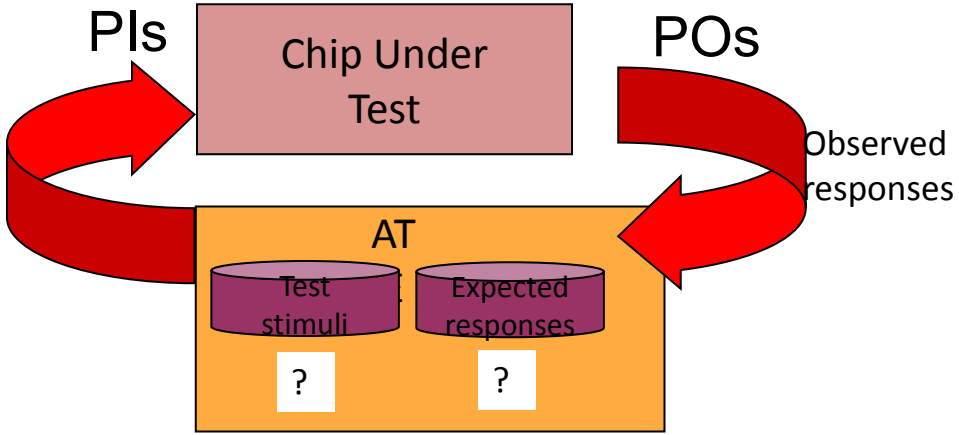
Rajendran, Jeyavijayan et al. ,"Security Analysis of Logic Obfuscation”, DAC 2012

Outline

Sensitization attack	SAT attack	Test-data mining attack
Threat Model Encrypted netlist Functional IC	**Threat Model** Encrypted netlist Functional IC	**Threat Model** Encrypted netlist Test data
Attack method Sensitize individual key bits to primary outputs	**Attack method** Eliminate incorrect keys using “distinguishing input patterns”	**Attack method** Use test data to extract the secret key for pre-test activation
Defense Strong Logic Encryption	**Defense** SARLock	**Defense** Post-test activation

1. Rajendran, Jeyavijayan et al. "Security analysis of logic obfuscation”, DAC 2012
2. Yasin et al. “Activation of Logic Encrypted Chips: Pre-Test or Post-Test?.”, DATE 2016
3. Subramanyan et al. “Evaluating the Security of Logic Encryption Algorithms”, HOST 2015

Manufacturing test

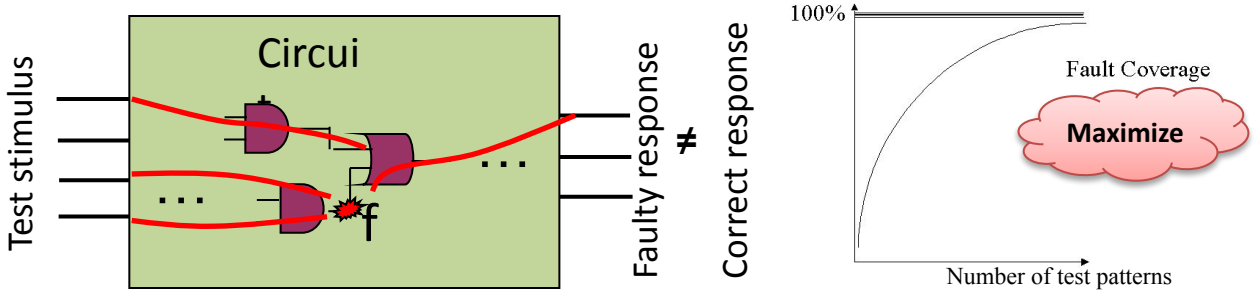


- Each IC passes through a manufacturing test
  - IC functional or defective?
- Automatic test equipment (ATE) is used
- Expected and observed responses are compared pass/fail

Automatic test pattern generation

Goal: Minimally sized subset of input vectors that capture *as many defects as possible*

→ **Fault:** Logic level abstraction for physical defects



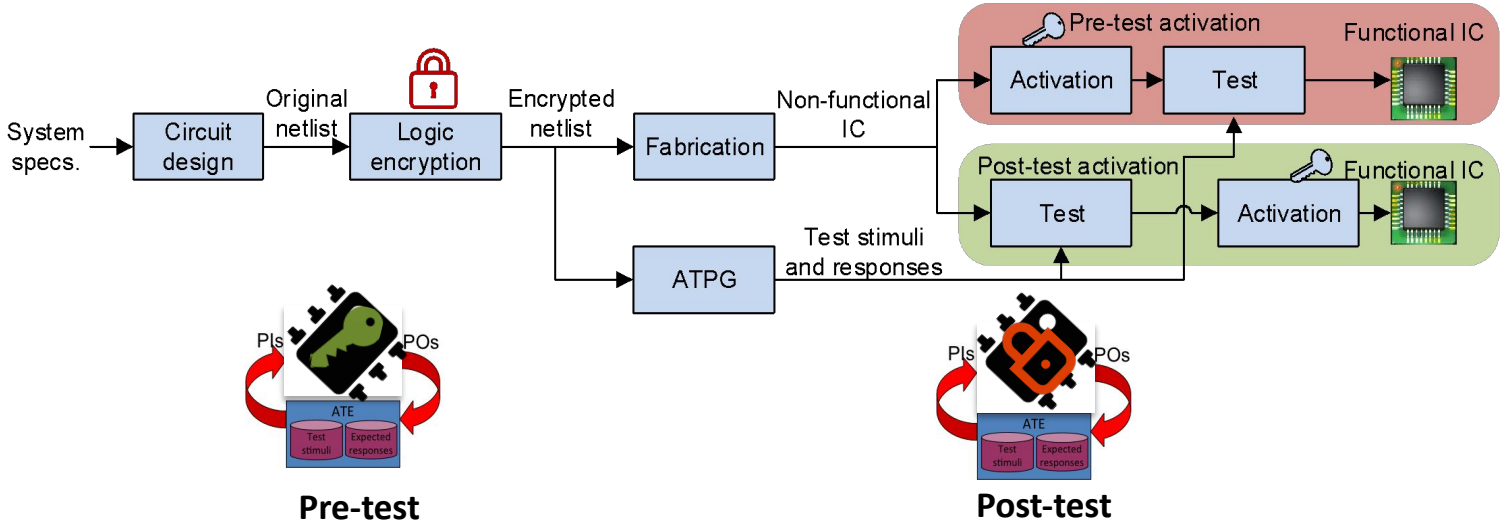
Logic tracing for:

- Activating the fault
- Propagating the fault effect

Test quality measured in terms of fault coverage:

- Percentage of faults detected by the test data

Activation of logic encrypted ICs



IC activation **prior** to test

Test conducted w/ correct key in place

ATPG constraint: correct key

IC activation **post** test

Test conducted w/o correct key loaded

ATPG **oblivious** to correct key

Evolving trust in the IC test

Traditional IC

- Conducted by the foundry itself (foundry trustworthy??)
- **Trusted** test personnel

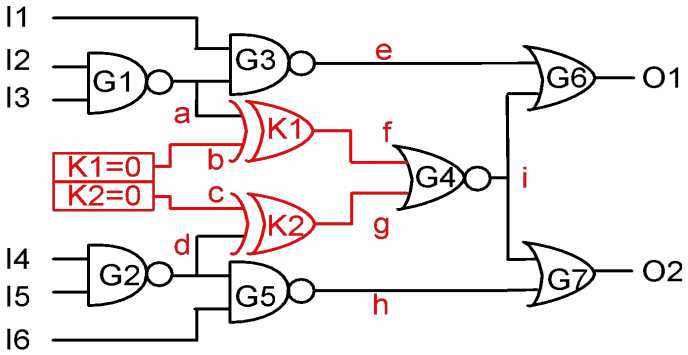
IC test

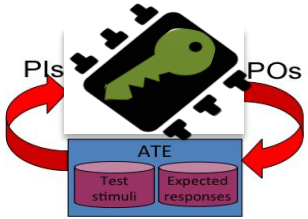
- **Outsourced** assembly & test[1] (OSAT) companies
- **Untrusted** test personnel

- Test personnel have access to:
  - Test patterns (test stimuli and expected responses)
  - Automatic test equipment
  - ICs to be tested
- Untrusted test facilities **IC secret exposed!**

1. Khadpe, Subash. "Outsourced semiconductor assembly and test: preparing for the next boom cycle, 2006-2008." *Chip Scale Review Magazine, April* (2005).

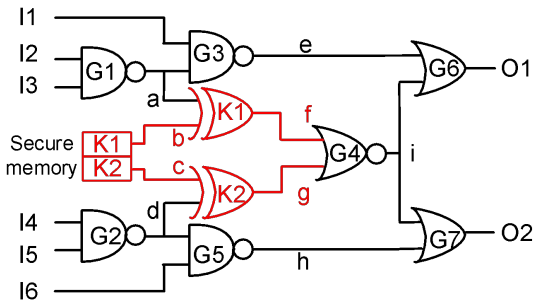
Pre-test Activation





- Manufacturing test is conducted with the correct key in place
- ATPG conducted: constraint = correct key

Pre-test Activation: Impact on security



Stimulus (T)	Response (Γ)
011001	10
101010	01
101111	01
011101	10
111010	11
000111	11
110001	00
001011	10

- ATPG constraint: correct key
  - Test data embeds info that can infer correct key
- Attack: Analyze test data to break logic enc.

Attack 3: Test data mining attack

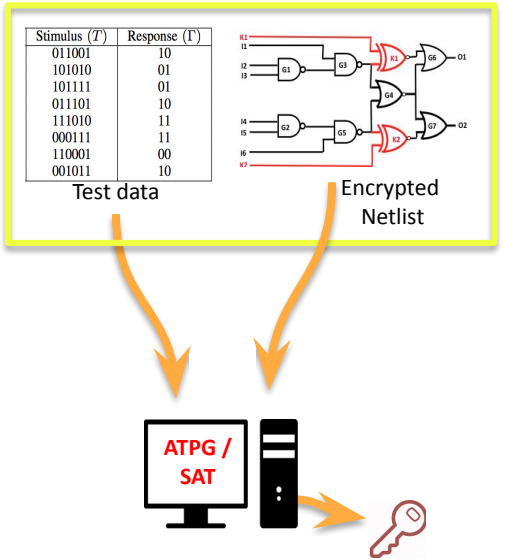
Goal: Determine secret key of logic enc.

Attacker has:

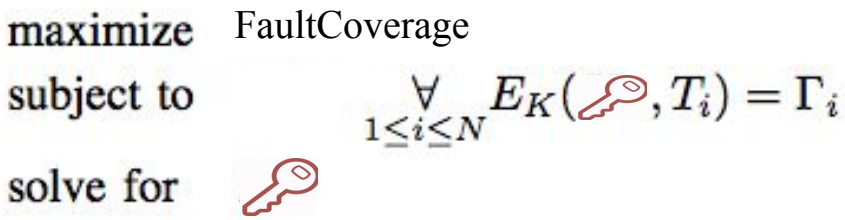
- An encrypted netlist E_K
- Test stimuli T and responses Γ.

Attack:

- Applies the test stimuli T and responses Γ as constraints
- Search for the key:
  - that satisfies the constraints
  - that **maximizes** the fault coverage



Test data mining attack



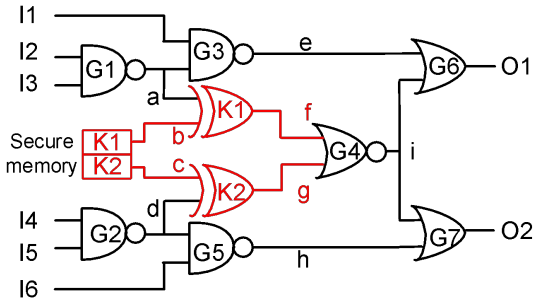
- System of Boolean equations can be solved using ILP techniques
- ATPG can also be used
  - NP-Hard

Test data mining attack (cont.)

$$\begin{aligned} &\text{maximize} && \text{FaultCoverage} \\ &\text{subject to} && \underset{1 \le i \le N}{\forall} E_K(\text{🔑}, T_i) = \Gamma_i \\ &\text{solve for} && \text{🔑} \end{aligned}$$

- Rationale:
- Test pattern generation: maximize fault coverage
  - correct key applied as constraint
- Attack: correct key maximizes fault coverage
  - test patterns applied as constraints

Test data mining attack (cont.)



Stimulus (T)	Response (Γ)
011001	10
101010	01
101111	01
011101	10
111010	11
000111	11
110001	00
001011	10

- Attack on the example design
- The only key that satisfies all the test pattern constraints is “00”
- Attack successful!

Experimental setup

- Results of the proposed attack
- Impact of activation models on test parameters
- Logic encryption with 32 key gates
- Three logic encryption techniques:
  - RLE, FLE, SLE
- Tetramax ATPG used for ATPG as well as launching the attack

Attack success

- The proposed attack is **100% successful against all logic encryption techniques**
- Both the hill climbing and the proposed attack are **100% successful** against RLE and FLE
  - 100,000 initial random keys are tried for the Hill climbing attack
- The hill climbing attack fails for certain SLE circuits,
  - when ratio of the # primary inputs to the # key inputs is small

SLE circuits

Benchmark	s298	s400	s444	s713	c432	s5378	c5315	c7552	s9234	s13207	s15850
Hill climbing	**No**	Yes	Yes	**No**	**No**	**No**	Yes	**No**	**No**	Yes	**No**
Proposed attack	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

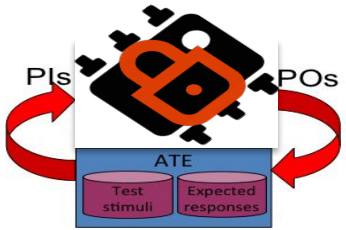
Attack success on partial test data

- The attack is **successful** with even **30% of test data** on SLE circuits

Benchmark	10%	20%	30%
c7552	No	No	Yes
s9234	Yes	Yes	Yes

Solution: Post-test activation

- Apply the key after test
- ATPG oblivious to correct key
- Test patterns do **not** embed information about correct key
- **Test data mining not applicable**



Key	Stimulus T	Response Γ
10	011011	11
01	111011	10
01	100110	11
11	101110	01
00	101110	01
01	101111	11
01	000101	10
01	111111	11
01	110000	01

Assessments - Week 8

Lab4 - Logic Locking

Lab4 - Logic Locking

- You get: locked and unlocked bitstreams
- locked netlist
- You need to use the unlocked bitstream + locked netlist
  - Calculate truth table
  - Identify DIPs
  - Use a SAT solver (logictools.org)
  - Identify the key!
- Bonus goal: Identify the key automatically using MicroPython

More details in the lab instructions!

Due Week 9 Friday