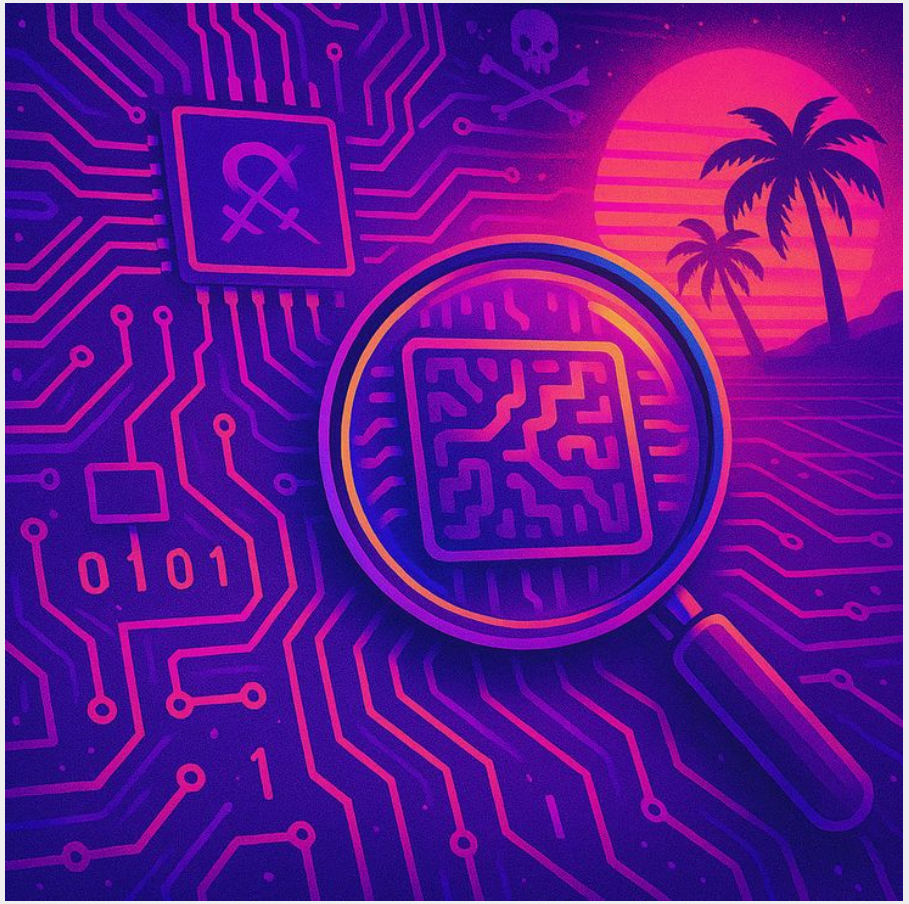
Hardware Security Week 9 - Piracy and Security: IC Camouflage



26T1

Hammond Pearce

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

RECAP

1. Why do we want to protect IPs?
2. How could an end-user attack an IP? How about a foundry?
3. What is the basis of Logic Locking?
4. How is Logic Locking defeated?

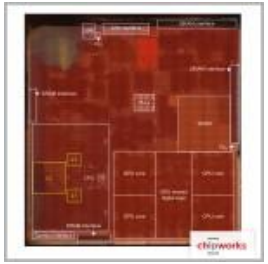
RECAP: Reverse Engineering (IP Theft?)

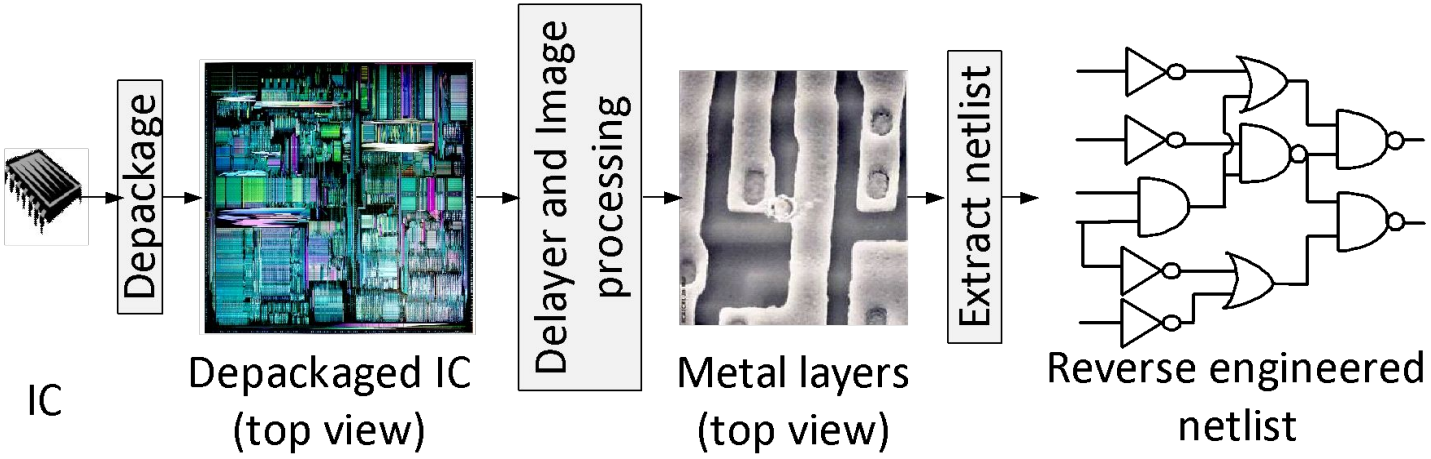
EE Times

System and IC teardowns become critical 'business intelligence'

chipworks

INSIDE THE NEW iPhone 6 and iPhone 6 Plus





RECAP: Reverse Engineering (IP Theft?)

- Identify device technology, functionality, design
- Can identify:
  - Piracy (did you use circuits not belonging to you?)
  - Trojan insertion (does this circuit make sense here?)
- But also enables:
  - Piracy (can steal the designs that have been reversed)
  - Trojan design (can identify sites to add malicious changes)

RECAP: Reverse Engineering Examples

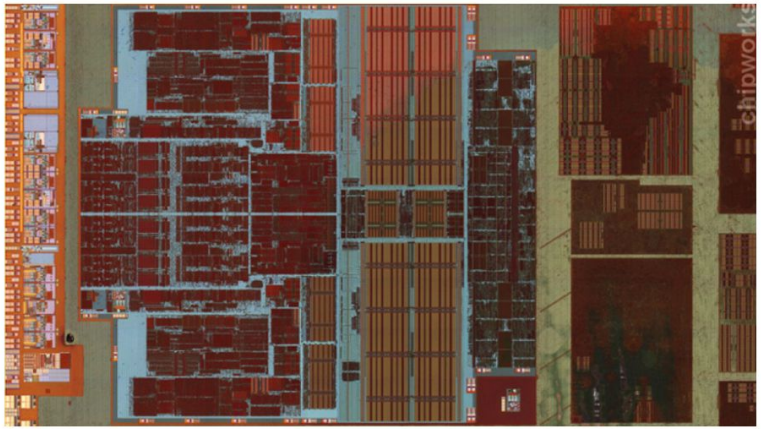
EXTREMETECH Aerospace Cars Computing Defense Electronics Energy Gaming Phones Science Security

Home > Computing

iPhone 5 A6 SoC reverse engineered, reveals rare hand-made custom CPU, and tri-core GPU

Apple still hasn't said a word about the new A6 SoC within the iPhone 5, but no matter: Chipworks has now completed its initial analysis of the A6, and the results are very interesting indeed. The A6 features a custom, in-house, laid-out-by-hand dual-core design (pictured above) that is neither a Cortex-A9 or A15, and a tri-core GPU.

By Sebastian Anthony September 25, 2012



DIGITIMES asia TECH REGIONS RESEARCH OPINIONS FINANCE BIZ FOCUS

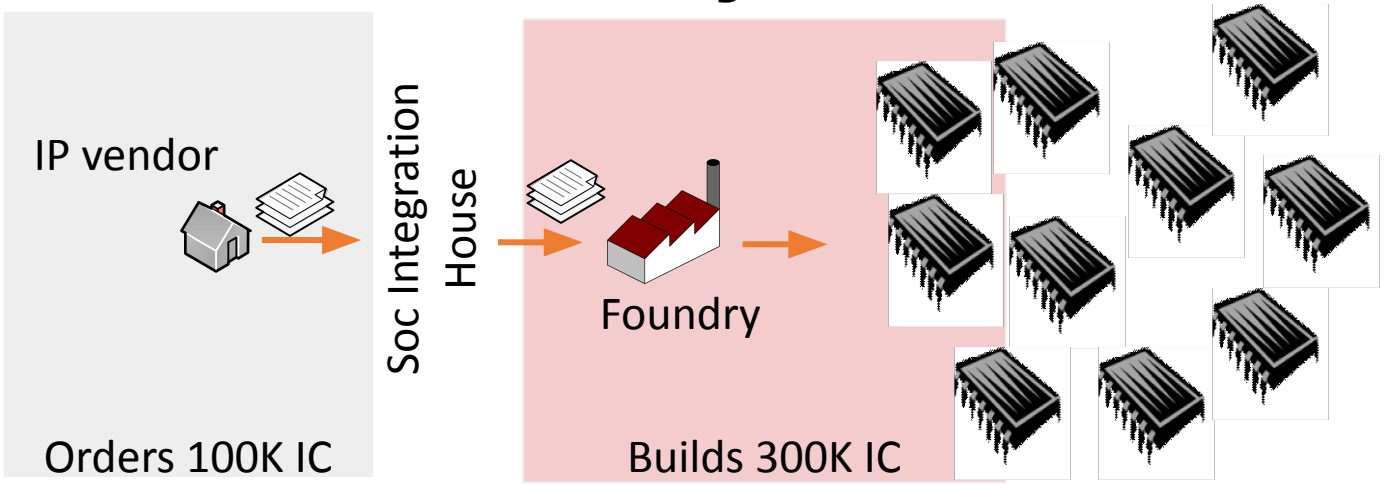
Academia: China's astonishing reverse engineering capabilities spur steep rise in US strategic concerns

Bryan Chuang, Taipei; Willis Ke, DIGITIMES Asia | Thursday 12 October 2023 | 0 Like 0

PHOTONICS spectra®

Using Reverse Engineering to Discover Patent Infringement
By Julia Elvidge, President, Chipworks

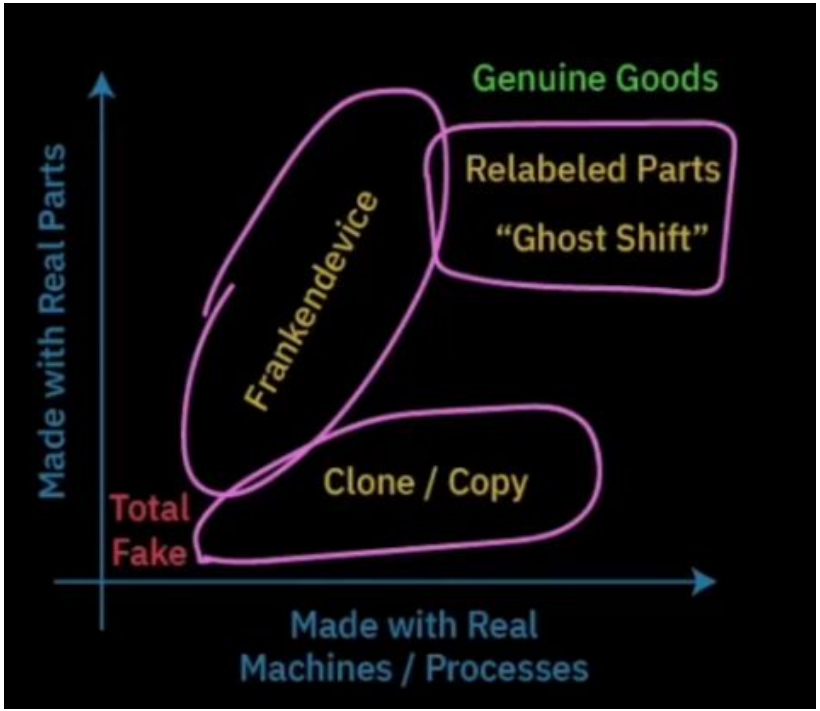
RECAP: IC/IP Piracy and “Overbuilding”



- Steal & claim ownership of an IC
- Scenarios:
  - Malicious SoC integration house
  - Malicious foundry

Fake vs Real is a Spectrum

Bunnie Huang, BH Asia 2025



Made with Real Parts
Vs.
Made with Real Machines/Processes

A photo from Shenzhen, 2AM: Factory workers produce reels of "fake" USB connectors

Last week:

Prevent a malicious factory/user from reverse engineering an IC

		User	
		Trusted	Untrusted
Foundry	Trusted		
	Untrusted		LOGIC LOCKING

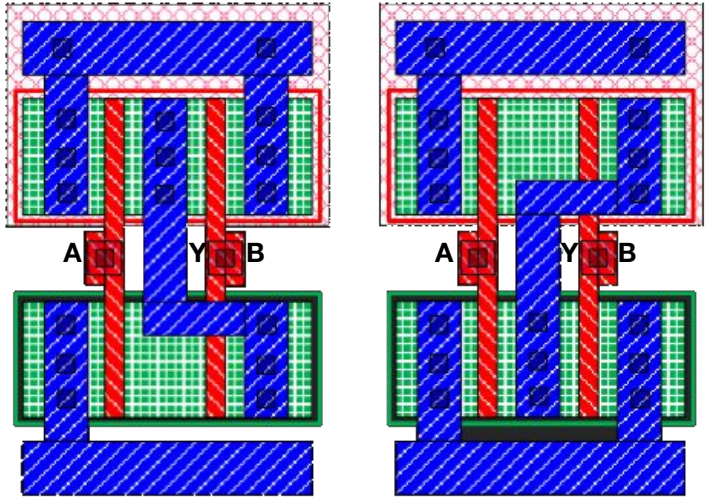
IC Camouflage:

Prevent a malicious user from reverse engineering an IC

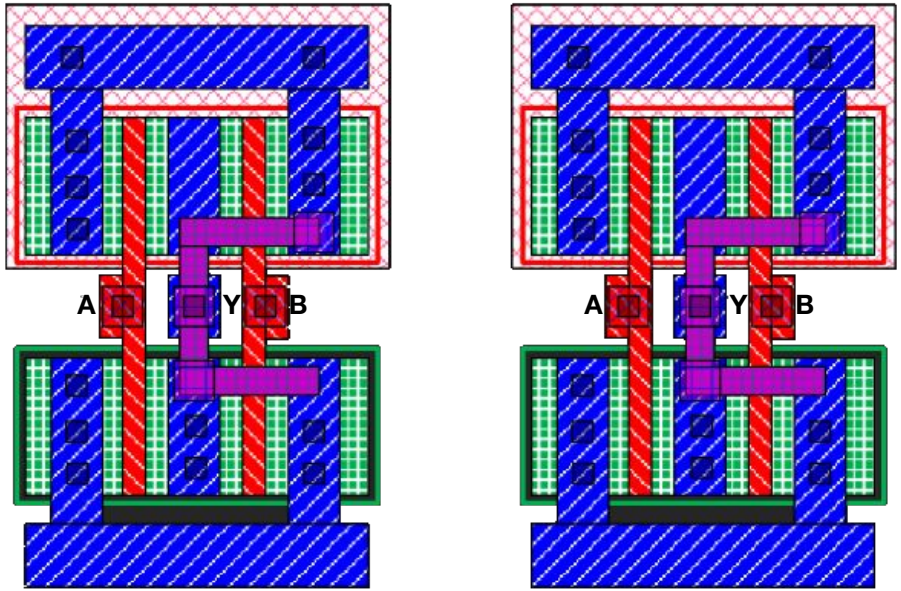
		User	
		Trusted	Untrusted
Foundry	Trusted		IC Camouflage
	Untrusted		LOGIC LOCKING

SypherMedia, “Syphermedia library circuit camouflage technology,” http://www.smi.tv/solutions.htm.

Design camouflaging?



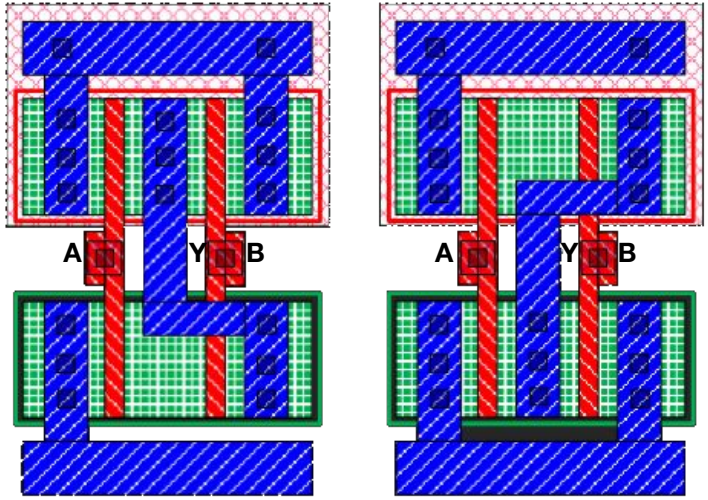
NAND NOR



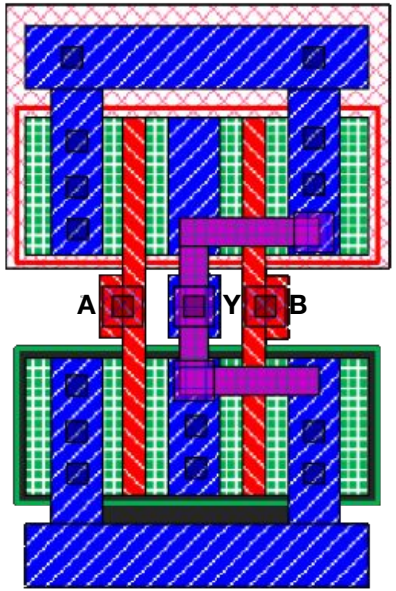
Camo. NAND Camo. NOR

SypherMedia, “Syphermedia library circuit camouflage technology,” http://www.smi.tv/solutions.htm.

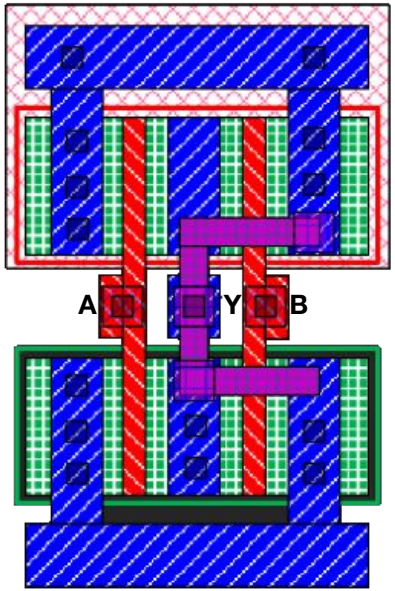
Design camouflaging?



Layout pattern → functionality
✗ (Easy to reverse engineer)
✔ Low power, delay, area



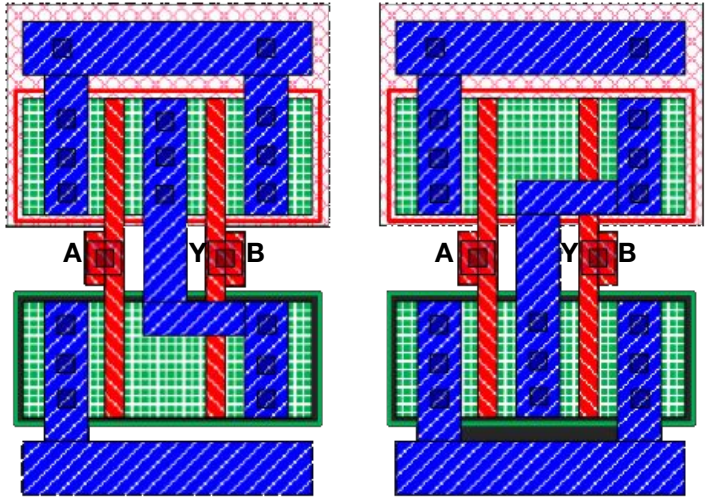
Camo. NAND



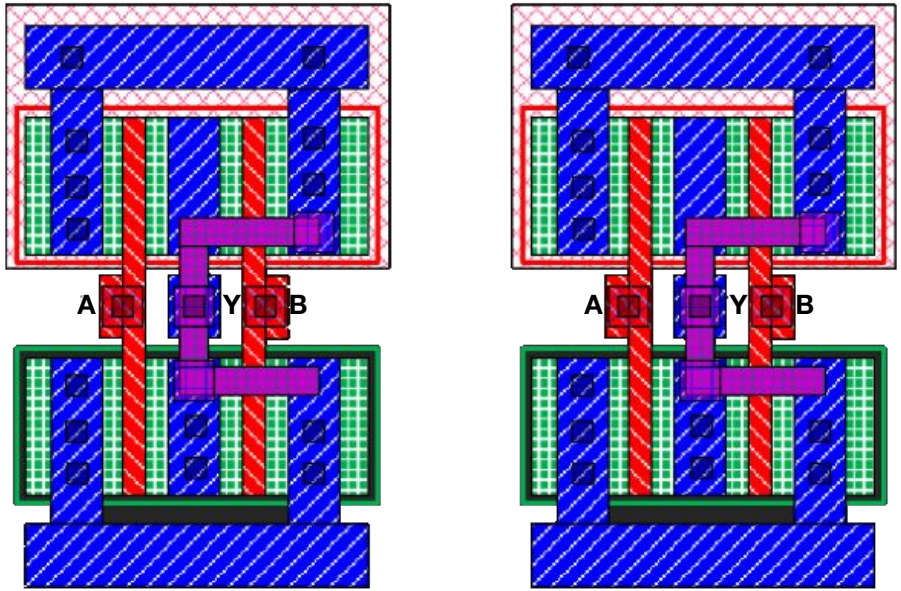
Camo. NOR

SypherMedia, “Syphermedia library circuit camouflage technology,” http://www.smi.tv/solutions.htm.

Design camouflaging?



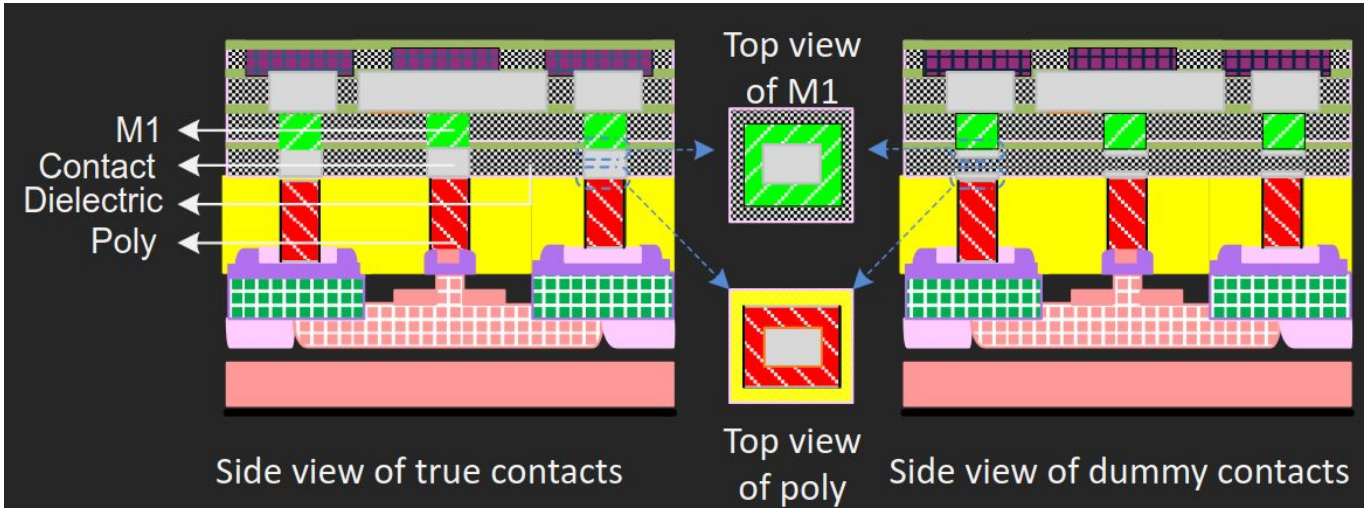
NAND NOR



Camo. NAND Camo. NOR

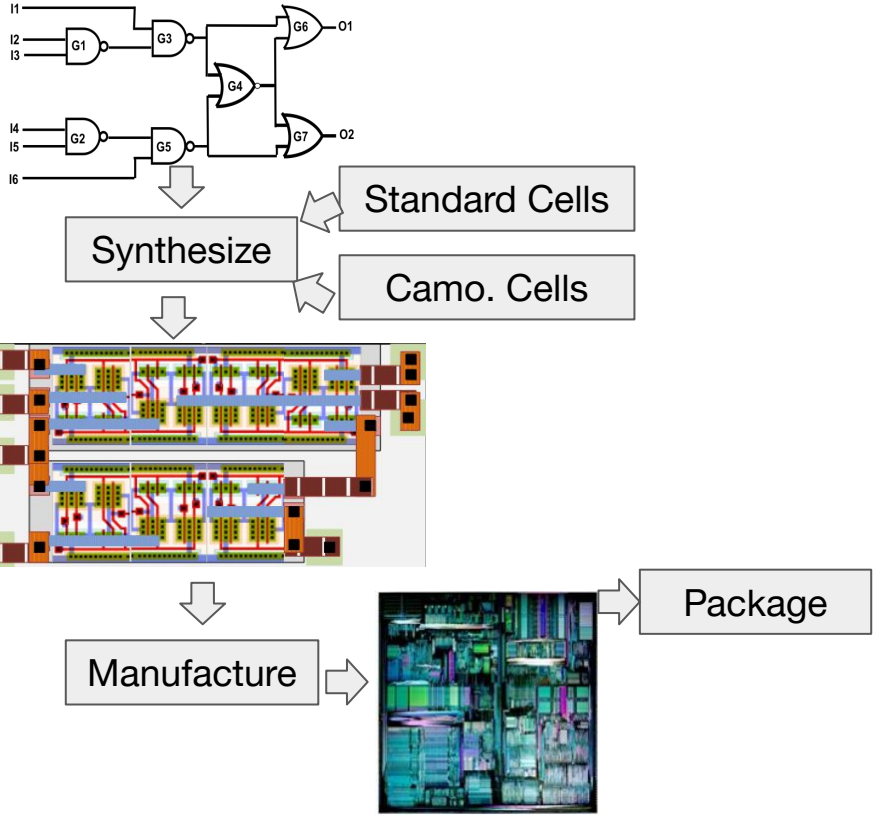
Layout pattern ↛ functionality
✔ (Hard to reverse engineer)
✗ High power, delay, area

Dummy contacts

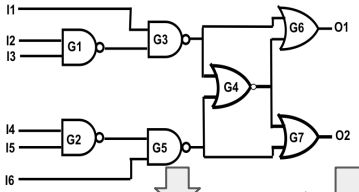


- True and dummy contacts identical from top view
  - Different electrically
- Proprietary technology of Syphermedia International

Design Flow


I1
I2
I3
I4
I5
I6
G1
G2
G3
G4
G5
G6
G7
O1
O2
Standard Cells
Synthesize
Camo. Cells
Manufacture
Package

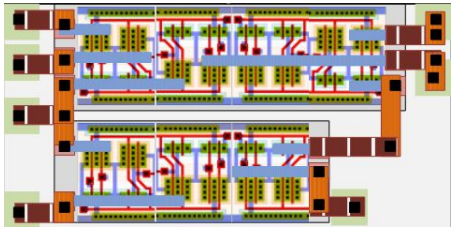
Design Flow



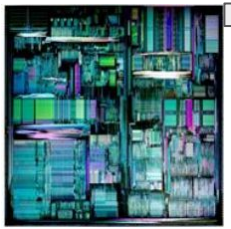
Synthesize

Standard Cells

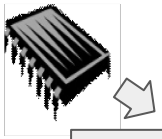
Camo. Cells



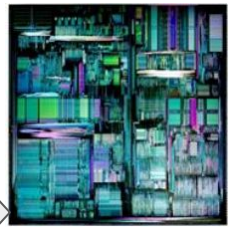
Manufacture



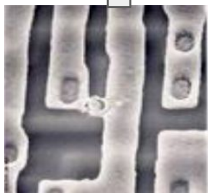
Package



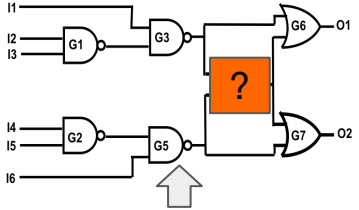
Depackage



Delayering & Image Processing

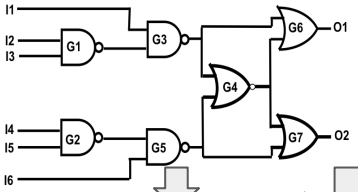


Analyze



Design Flow

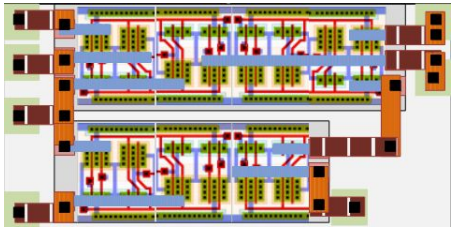
Defender:
Camo. gates → Security vs. PPA
Attacker: Camo. gate function?



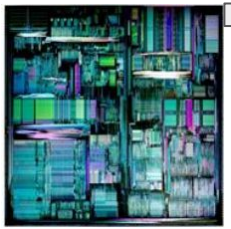
Synthesize

Standard Cells

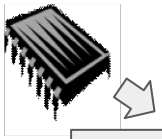
Camo. Cells



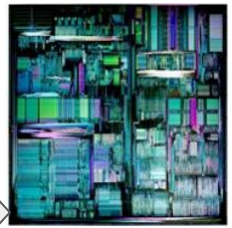
Manufacture



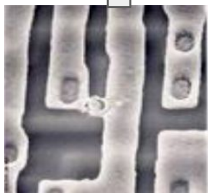
Package



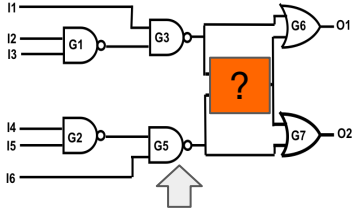
Depackage



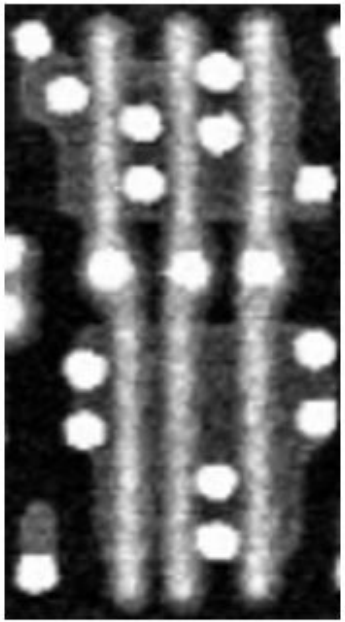
Delayering & Image Processing

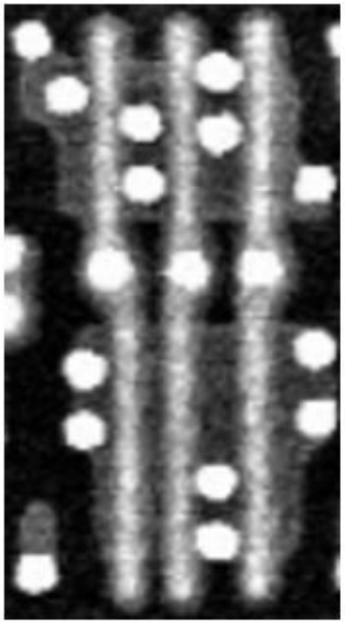


Analyze



Example - what's the difference here?





IC Camouflaging in the Market

- Feasible?
- Who uses it?

- To what extent? *>140M camouflaged ICs produced*
- Is it secure?

IC Camouflaging in the Market

- Feasible?
- Who uses it?

- To what extent? *>140M camouflaged ICs produced*
- Is it secure? …..?

Camouflage

- Higher power, delay, and area than normal gates
- Impractical to secure all gates
- But, it is in use! One estimate: >140M camouflaged ICs released

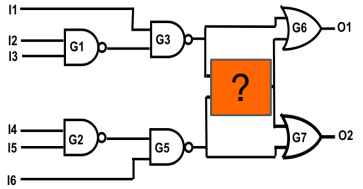
Metrics

- Attacker should not learn the outputs
  - Hamming distance between outputs of designs with correct & incorrect functionality assignments of camo. gates should be 50%
    - Entropy is maximum at 50%
    - 0% - no difference
    - 100% - outputs are complements
  - (Sound familiar??)

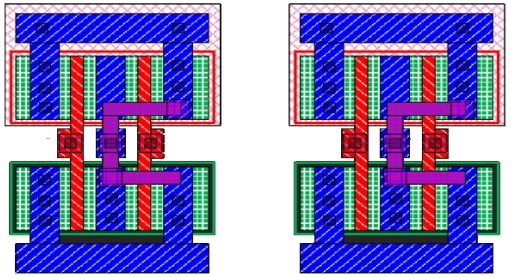
- Attacker should not learn the functionality of camo. Gates
  - With polynomial number of input and output pairs

Capabilities of an Attacker

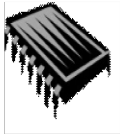
- Has the camouflaged netlist
  - By reverse engineering the IC
  - Knows which gates are camo'd
  - But not their functionality



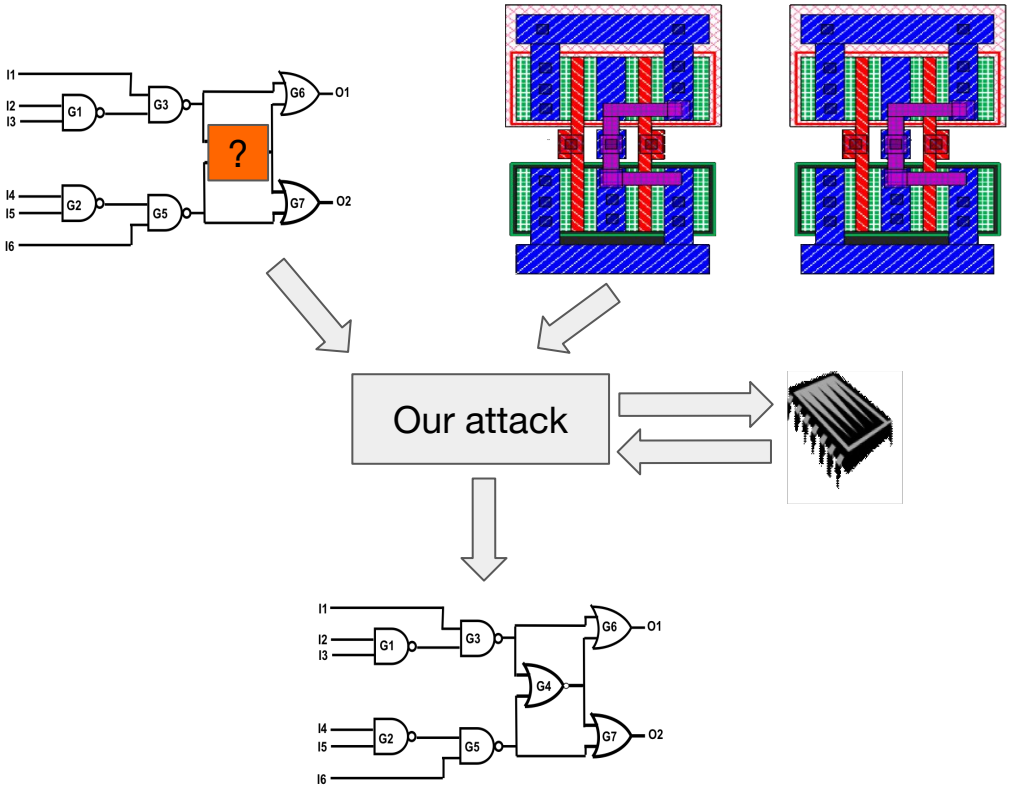
- Knows the camouflaged library
  - Got from companies



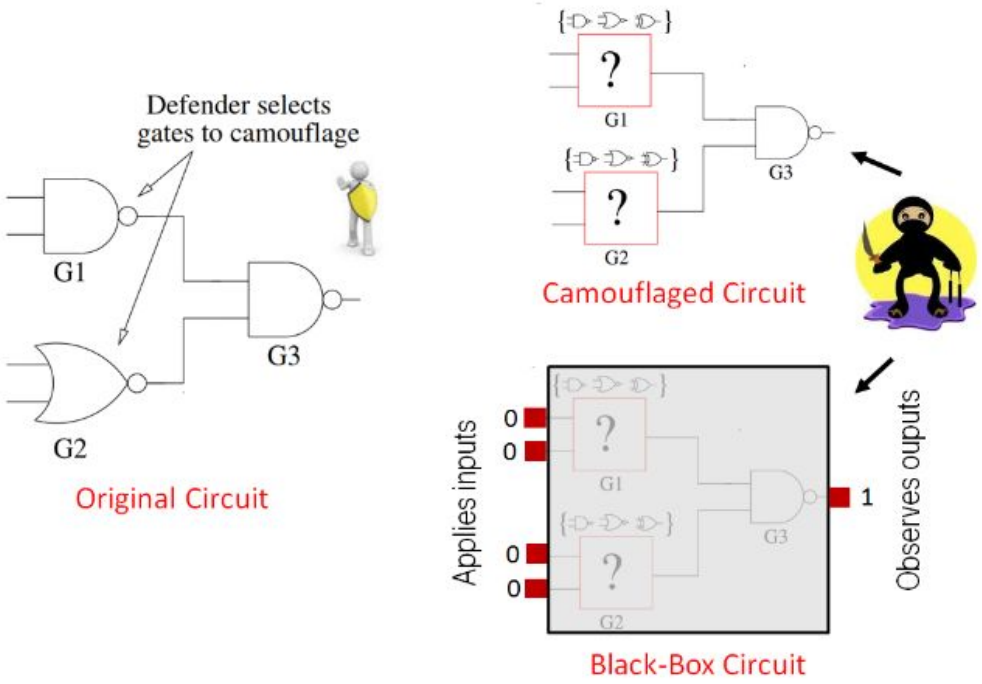
- Has a golden IC
  - Bought from the market



Attacking a Camouflaged IC



Defeating camouflaged circuits



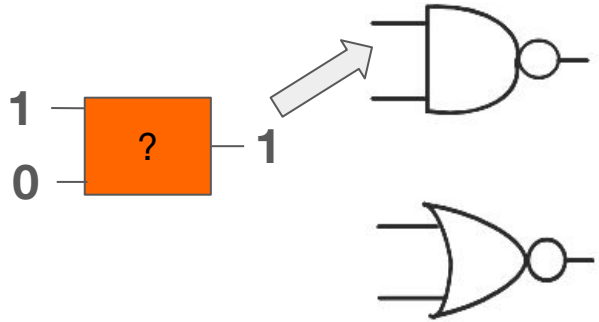
The attacker obtains two copies of the circuit:

Determines netlist with camouflage gates from first,

Tests *discriminating* inputs on second to quickly identify some gates,

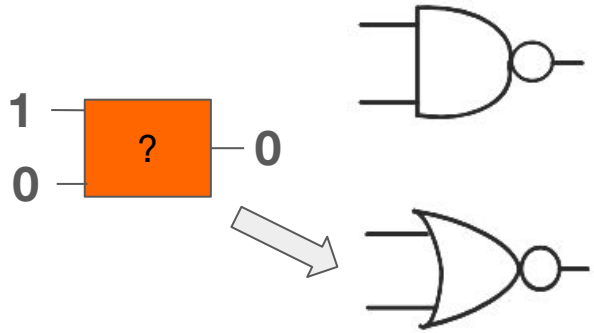
Then brute forces remaining unknown gates.

Attack: Basics



In	NAND	NOR
00	1	1
01	1	0
10	1	0
11	0	0

Attack: Basics



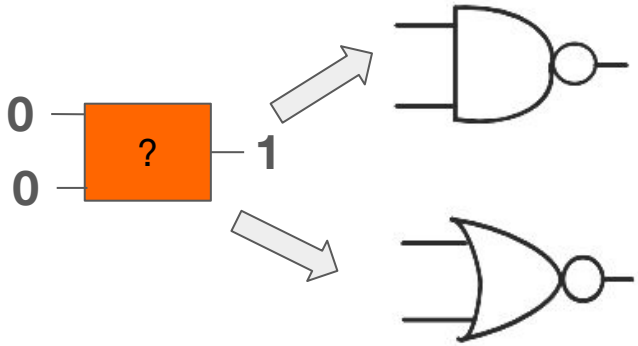
In	NAND	NOR
00	1	1
01	1	0
10	1	0
11	0	0

DP

Differentiating Patterns (DP)

- Different outputs for different functionalities
- Attacker uses on targeted camo. gate

Attack: Basics



In	NAND	NOR
00	1	1
01	1	0
10	1	0
11	0	0

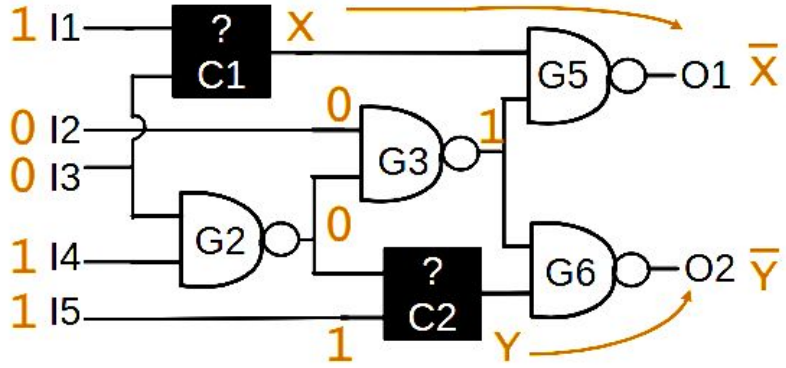
DP

CP

Common Patterns (CP)

- Same output regardless of functionalities
- Attacker mutes non-targeted camo. gates

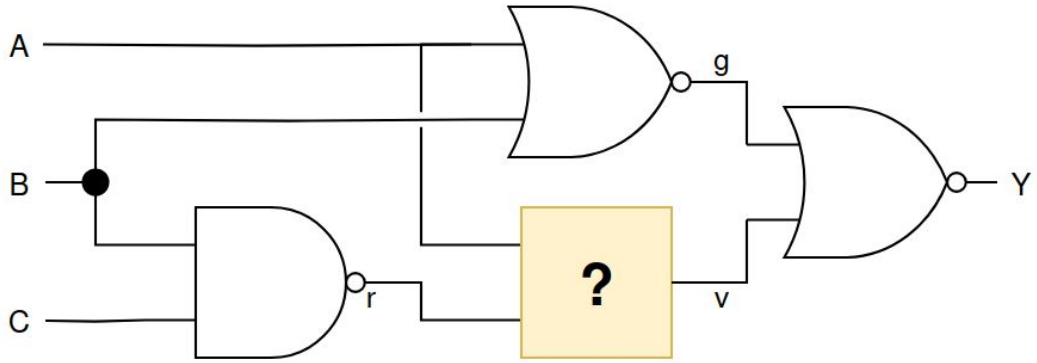
Attack: Isolated



Isolated camo. gate: no interference with any other camo. gate

- Justify its inputs to DP
- Sensitize (bijectively map) its output to a primary input

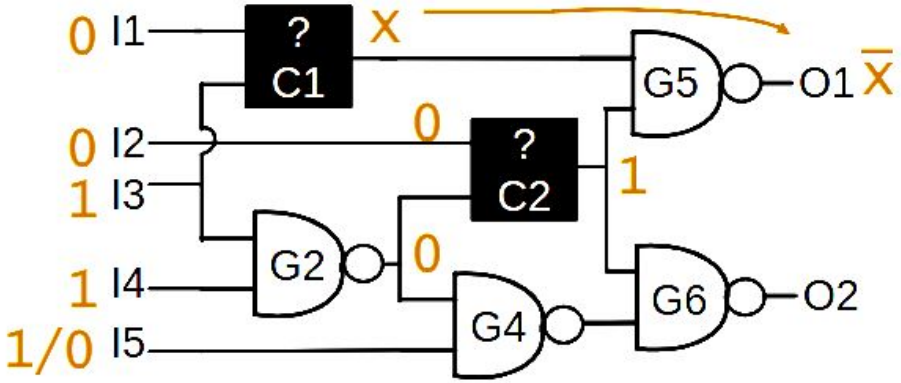
Class exercise: 5 mins



A	B	C	Y
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

- Is the “?” gate a “NAND” or a “NOR” gate?

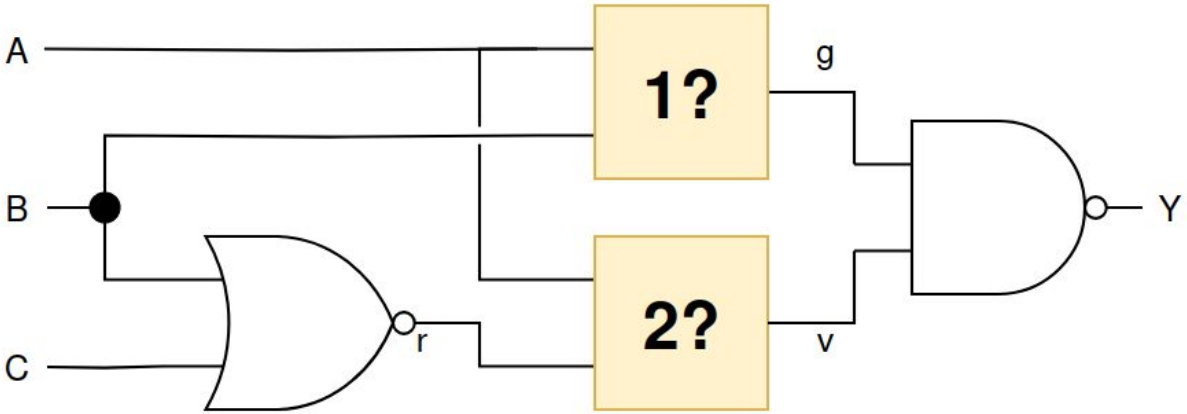
Attack: Resolvable & Interfering



Resolvable & Interfering camo. gates

- On each other's path or converge at some other gate
- Non-targeted camo. gates; bring CP to inputs
  - Target camo. gate:
  - Bring DP to inputs, sensitize output to a primary output

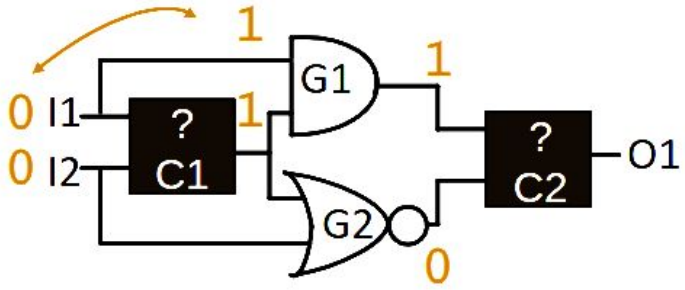
Class exercise: 5 mins



A	B	C	Y
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

- Is the “1?” gate a “NAND” or a “NOR” gate?
- Is the “2?” gate a “NAND” or a “NOR” gate?

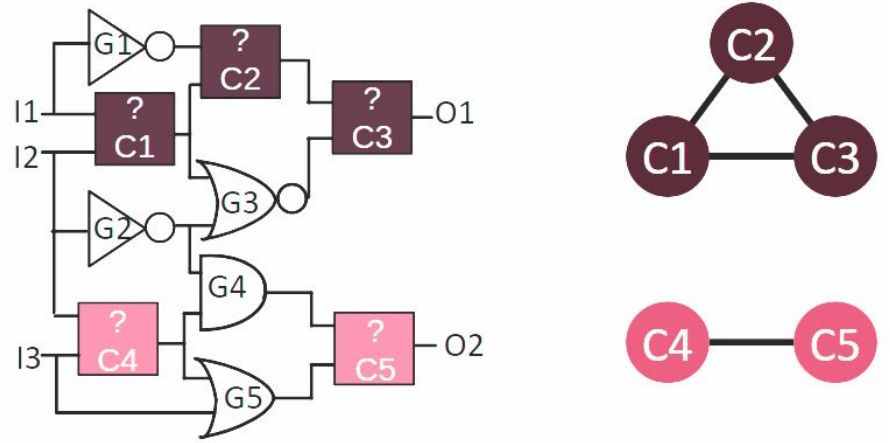
Attack: Non-resolvable & Interfering



Non-resolvable and interfering camo. gates:

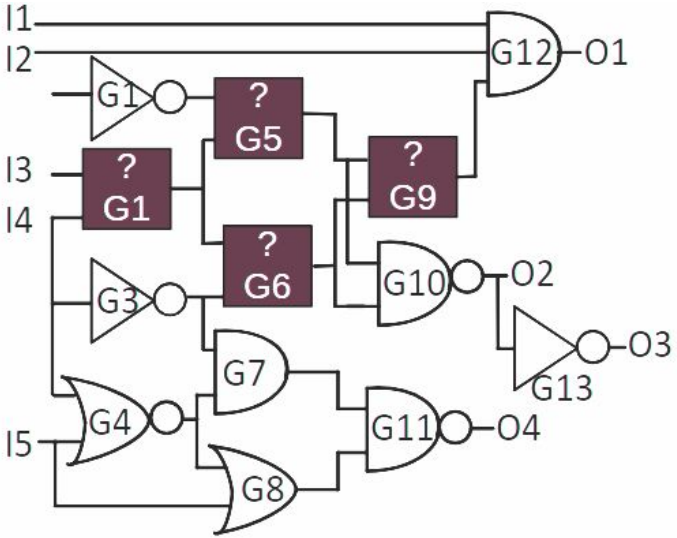
- Cannot be justified to DP and propagated to an output
- Needs brute force

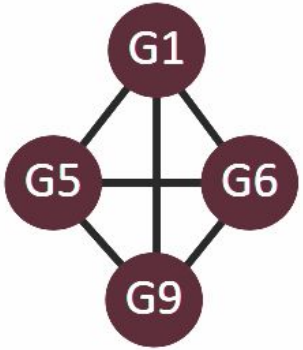
Metrics



- Interference graph:
  - Node ≈ Non-resolvable camo gate
  - Edges connect interfering gates
  - ↑ in # cliques → linear ↑ in brute force effort
  - ↑ in size of a clique → exponential ↑ in brute force effort
- Metric: Size of largest **clique** in the interference graph

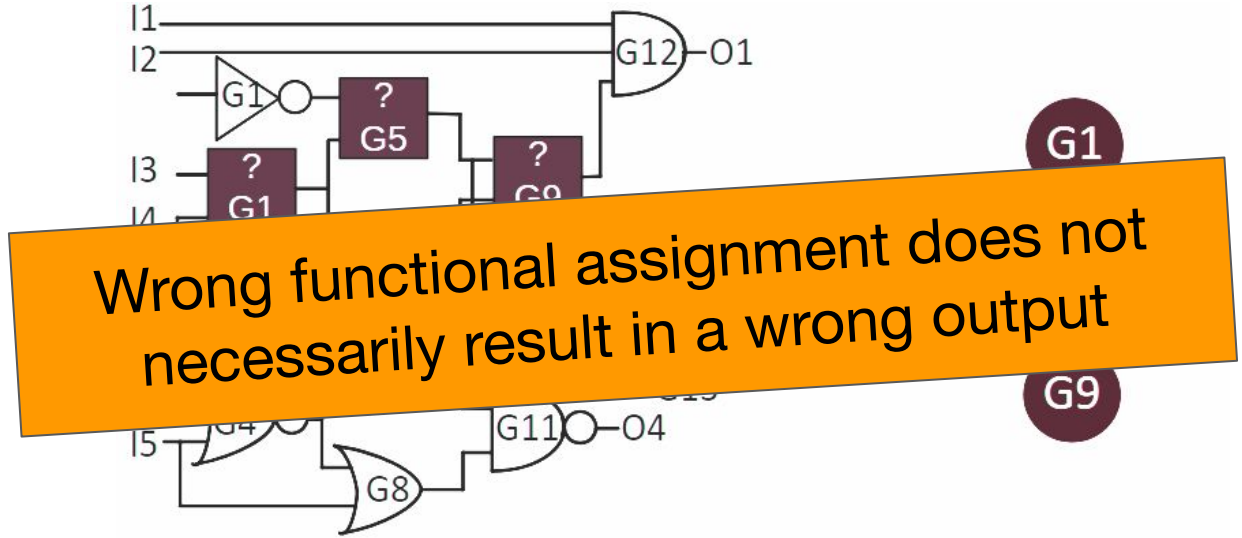
Gate selection





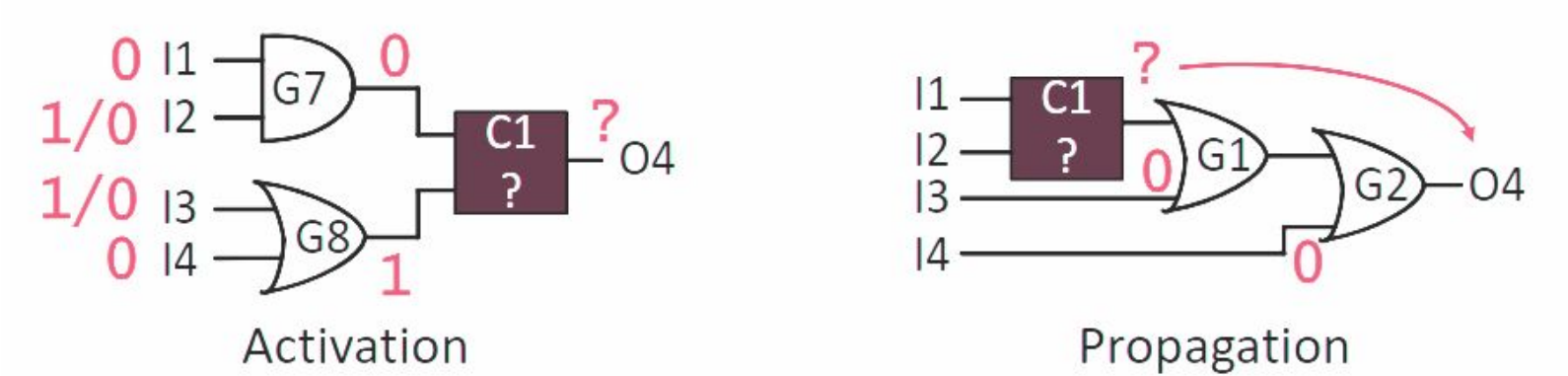
- At each iteration
  - Randomly select one of the non-resolvable gates w.r.t CamoGateList
  - Convert it to NAND/NOR if needed
  - Camouflage it and add it to CamoGateList
  - Repeat until “enough” gates are camouflaged

Gate selection



- At each iteration
  - Randomly select one of the non-resolvable gates w.r.t CamoGateList
  - Convert it to NAND/NOR if needed
  - Camouflage it and add it to CamoGateList
  - Repeat until “enough” gates are camouflaged

Output Corruptability



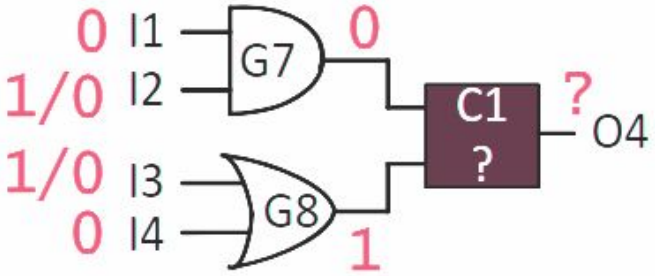
Activation:

- Need to activate ambiguity for max. no. of input patterns
- Ambiguity is activated only for DPs {01,10}

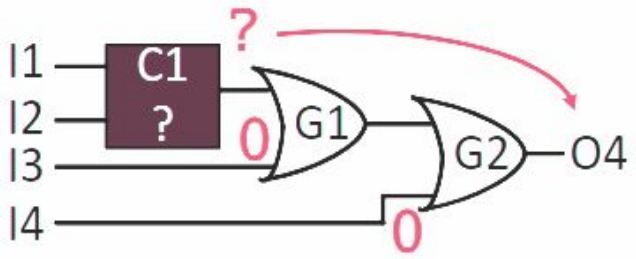
Propagation:

- Sensitize the ambiguity to max. no. of outputs
- Side-input of the other gates set to their non-controlling value

Output Corruptability



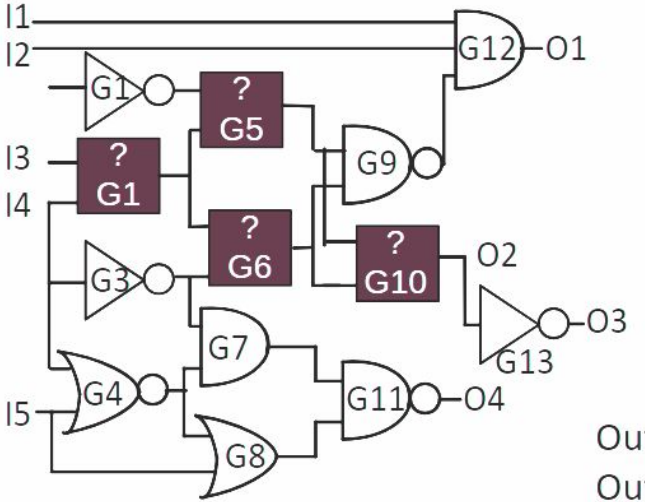
Activation

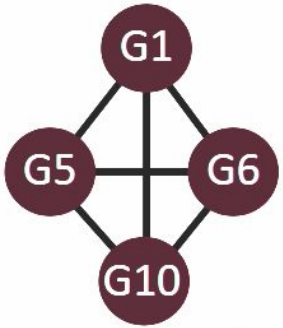


Propagation

$$\text{OutputCorruptability}_G = \sum_{\#\text{ of input patterns}} \#\text{ of DPs at G's input} \times \#\text{ of outputs with ambiguity}$$

Gate Selection 2.0





OutputCorruptability of G9 < OutputCorruptability of G10

- At each iteration
  - ~~Randomly select one of the non-resolvable gates w.r.t CamoGateList~~
  - Select non-resolvable gate with highest OutputCorruptability
  - Convert it to NAND/NOR if needed
  - Camouflage it and add it to CamoGateList
  - Repeat until “enough” gates are camouflaged

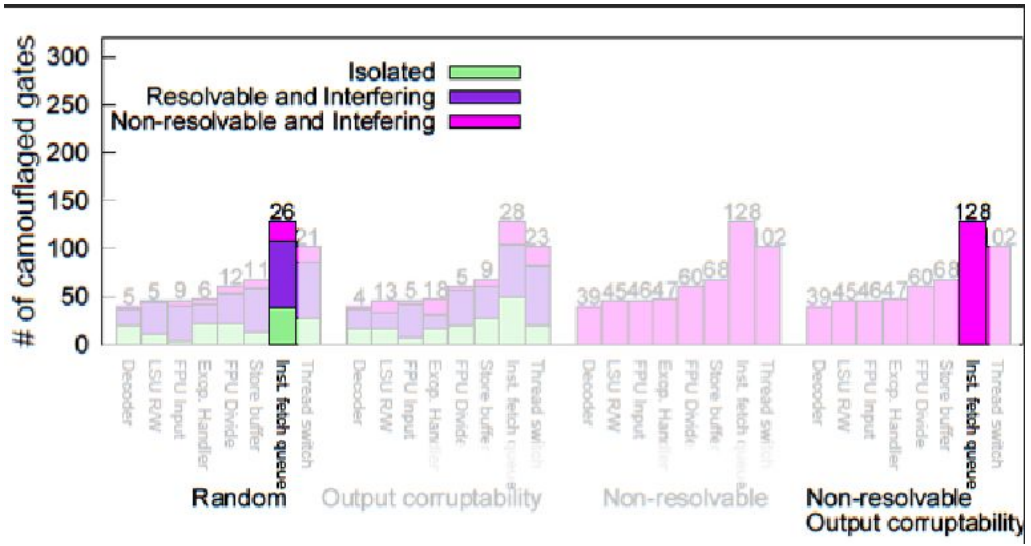
Results: Types of Camo. Gates

Random and Output-Corruptability

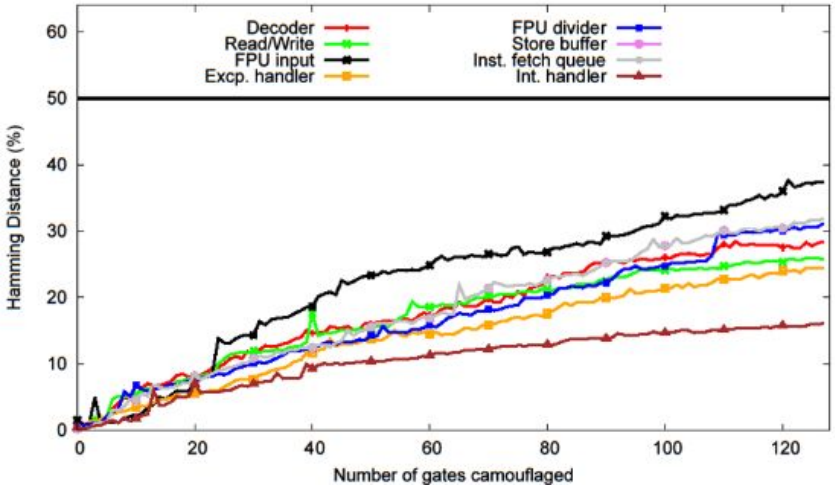
- Mostly Isolated / Resolvable gates
- Easy to break

Non-resolvable and
Non-resolvable+Output-Corruptability

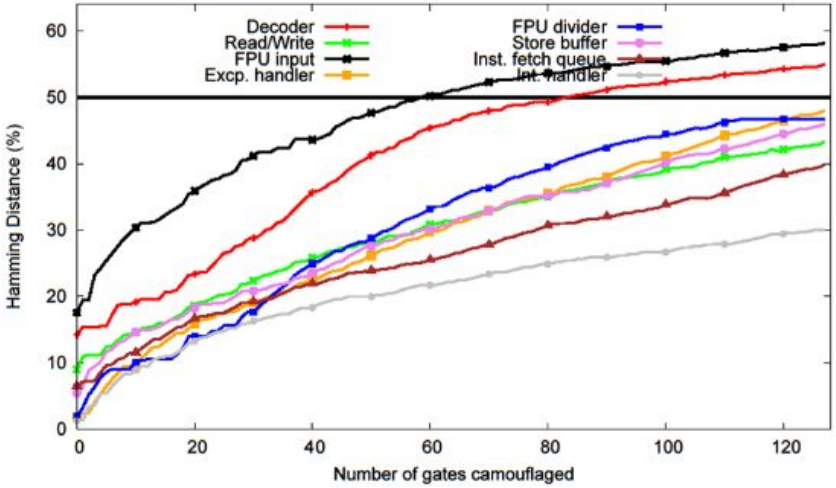
- Only non-resolvable gates
- Increases the effort for an attacker



Results: Hamming Distance



Random



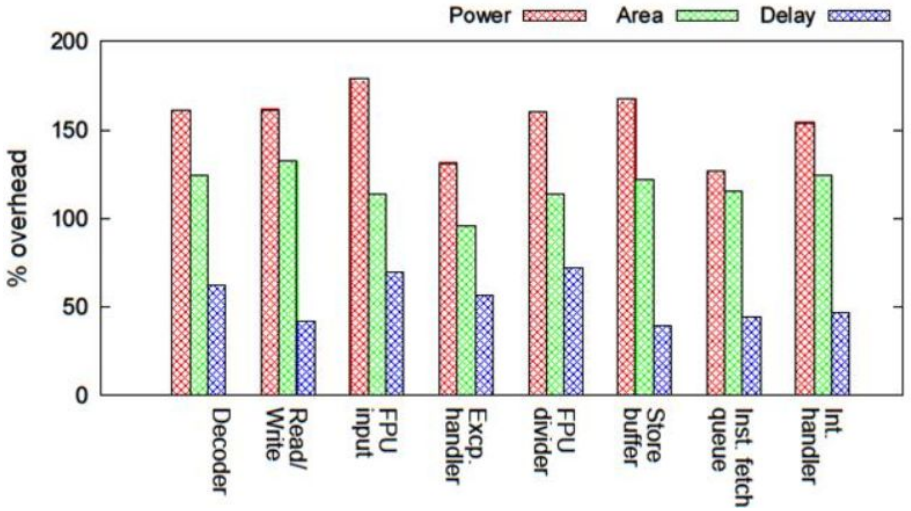
Output-Corruptability

Ideally, Hamming distance should be 50% max. ambiguity

- Random selection - Hamming distance <50%, Poor activation and propagation
- OutputCorruptability - Hamming distance ~50%, Activation and propagation are taken into account

Results: Overheads

Y. Alkabani et al., Active hardware metering for intellectual property protection and security, USENIX, 2007



Avg. power overhead is 105% when 5% of gates are camouflaged

- Camouflage cells are power hungry (~5X)
- Unused transistors causing leakage and switching power

However, controllers are tiny (≤1%) of overall design

- Camouflaging controllers lead to small overhead
- Without controllers, datapath is rendered useless

Unfortunately:

Integrated Circuit (IC) Decamouflaging: Reverse Engineering Camouflaged ICs within Minutes

Mohamed El Massad
University of Waterloo
melmassa@uwaterloo.ca

Siddharth Garg
New York University
sg175@nyu.edu

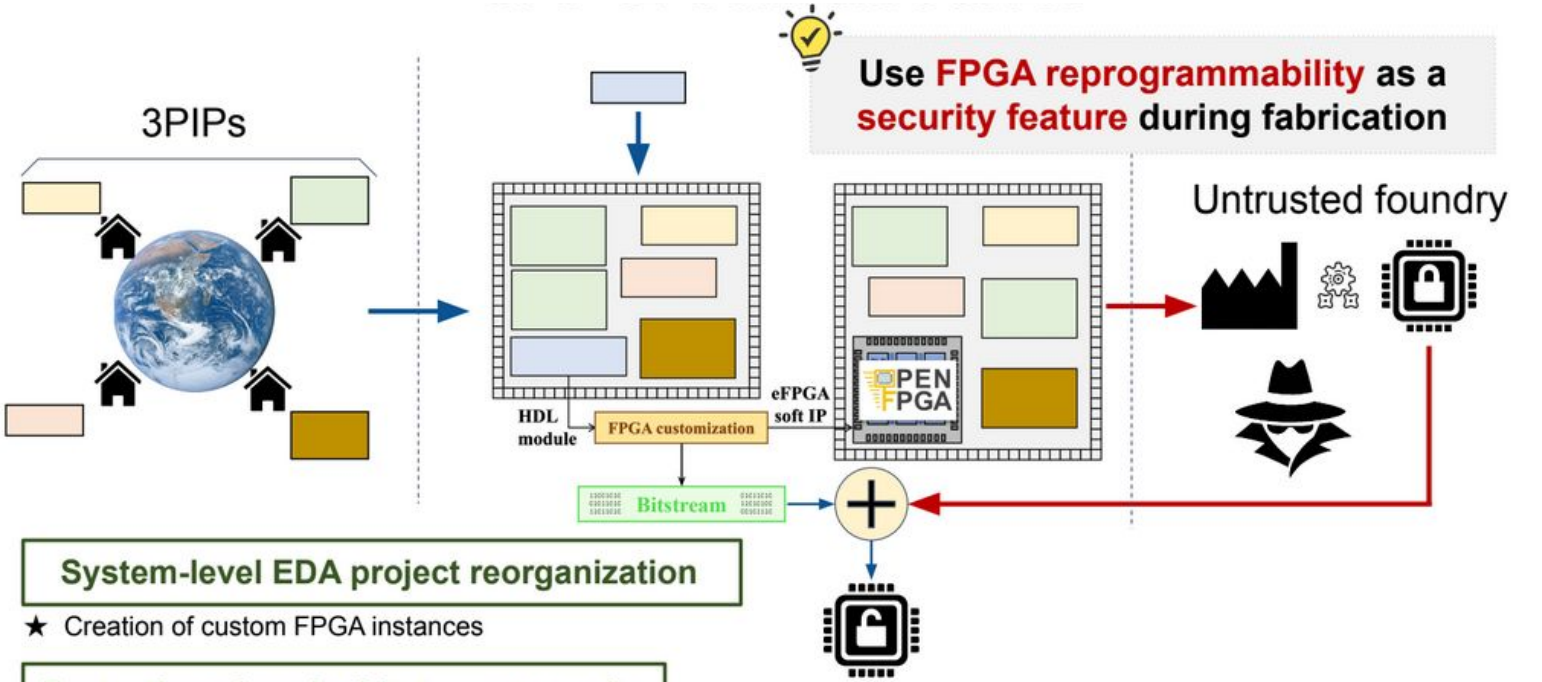
Mahesh V. Tripunitara
University of Waterloo
tripunit@uwaterloo.ca

[NDSS, 2017]

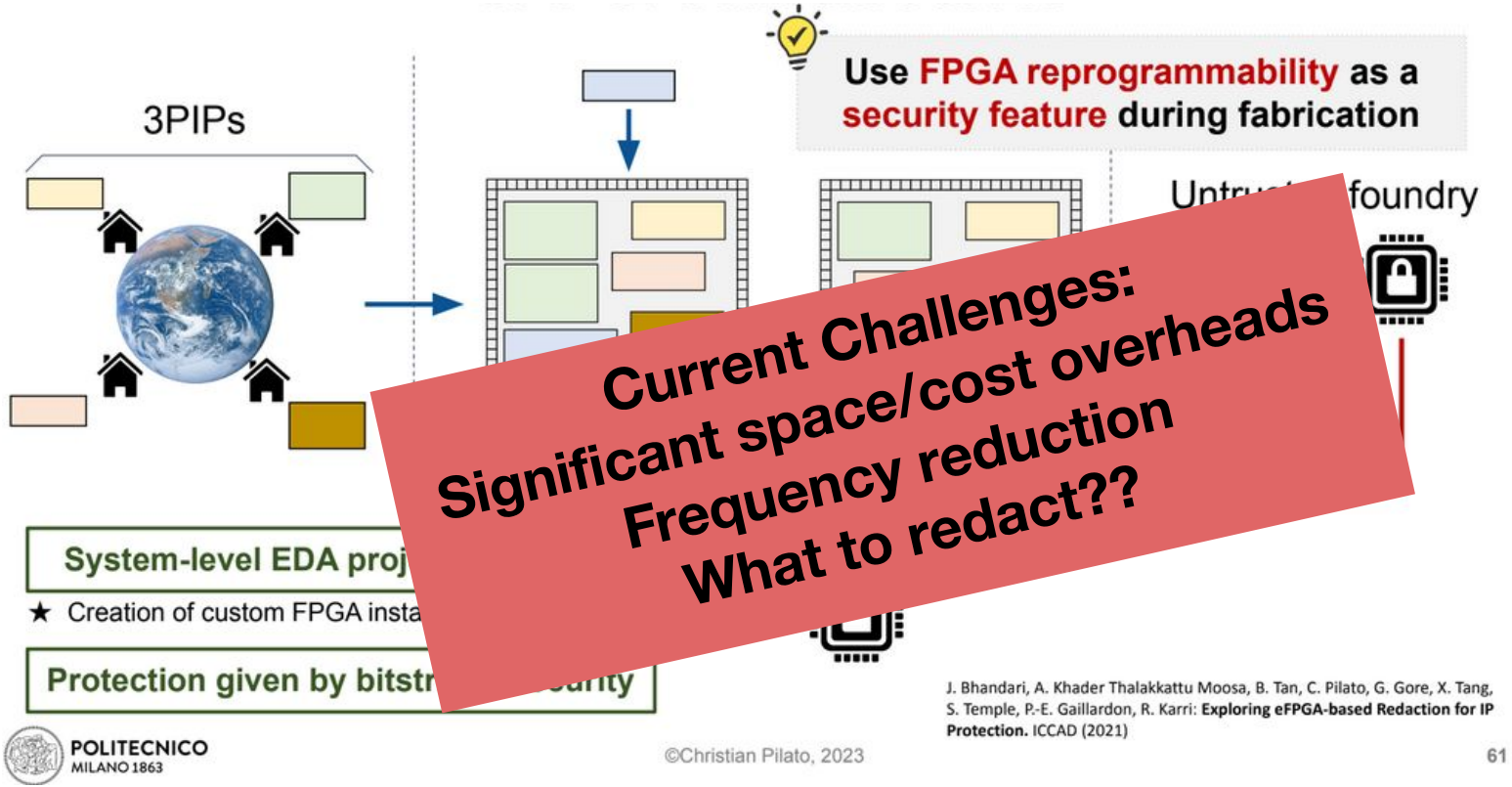
Damn.

Maybe future work can fix this?

Design obfuscation - eFPGAs?



Design obfuscation - eFPGAs?



Summary

- Camouflaging complicates reverse engineering by using look-alike cells
- Look-alikes can be designed using:
  - Dummy contacts, dummy transistors
  - Programmable cells, combination of low/high-Vt transistors
- Attacks similar to SAT-attack
  - Called “De-Camo” attack (principle is the same)
- What is missing?
  - Provably secure IC camouflaging

5 minute break

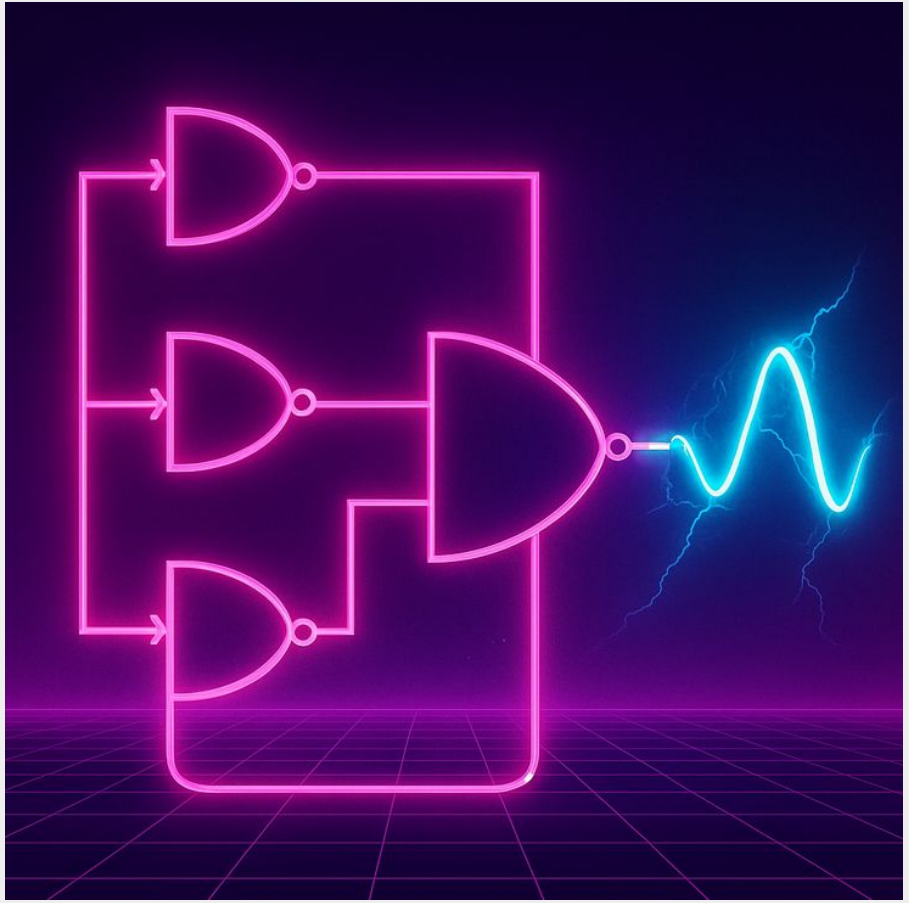
UNSW SYDNEY

Hardware Security Week 8 - Piracy and Security: Physically Unclonable Functions

26T1

Hammond Pearce

Slide material acknowledgements to Ramesh Karri, Farinaz Koushanfar, Steve Trimberger, Gang Qu



Physically Unclonable Function:

Prevent/detect unauthorized copies of hardware

Generate encryption keys securely “on the fly”

		User	
		Trusted	Untrusted
Foundry	Trusted		IC Camouflage
	Untrusted		LOGIC LOCKING, **PUF**

Security Challenges

- How to securely authenticate devices?
  - Keycards, RFIDs, mobile phones
  - Authentic vs. Counterfeits
- How to protect sensitive Intellectual Property (IP)?
  - Digital content
  - personal information
  - Software on mobile/embedded systems
  - Hardware I/P
- How to securely communicate with devices?
  - Private and authentic communication channels

Public key crypto-based authentication

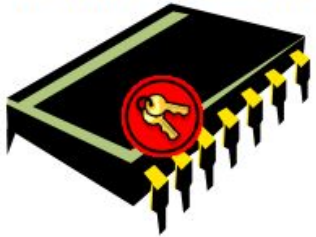
- Each IC needs to be unique
  - Embed unique secret key SK in on-chip non-volatile memory

- Use cryptography to authenticate IC
  - Verifier sends a randomly chosen number
  - IC signs number using its secret key
  - Verifier decrypts and ensures that IC has the secret key

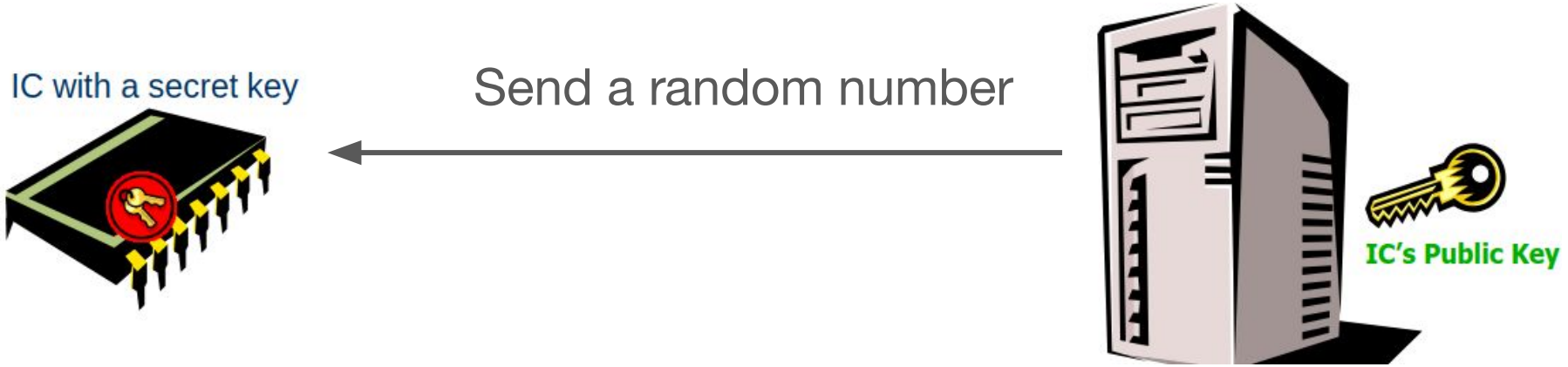
- Cryptography can protect IP, secure communication

Public key crypto-based authentication

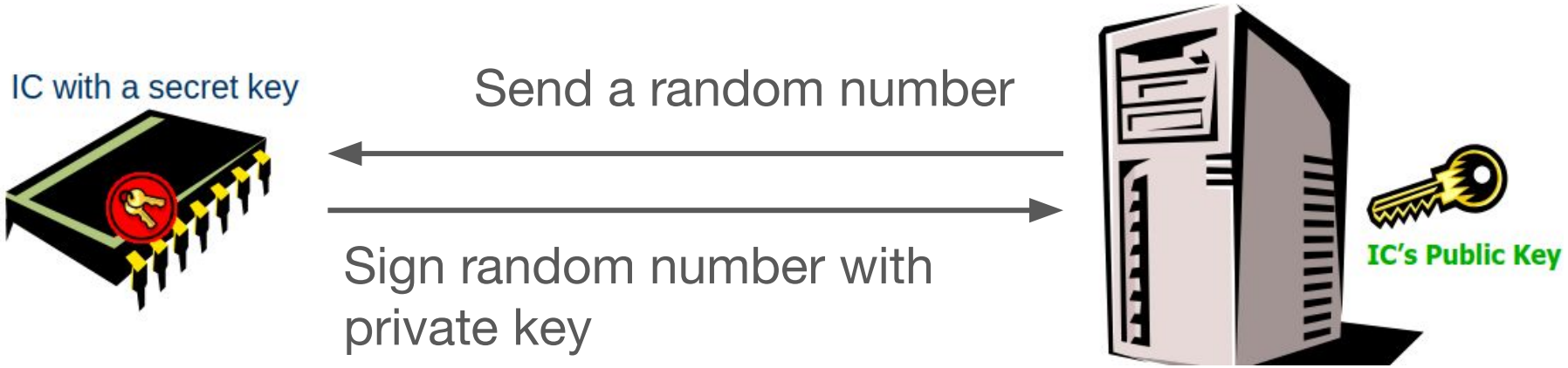
IC with a secret key



Public key crypto-based authentication



Public key crypto-based authentication



Only IC w/ key can generate a valid signature

BUT

- How can one generate and store secret keys on an IC in a **secure** and **inexpensive** way?
  - Adversaries may extract secret keys from non-volatile memory
  - Trusted party must embed and test keys in a secure location

- EEPROM adds additional complexity to manufacturing

- What if cryptography is NOT affordable or available?
  - Resource (e.g. power) constrained systems such as RFIDs
  - Commodity ICs such as FPGAs

The Problem

Storing digital information in a device in a way that is resistant to physical attack is difficult and expensive.

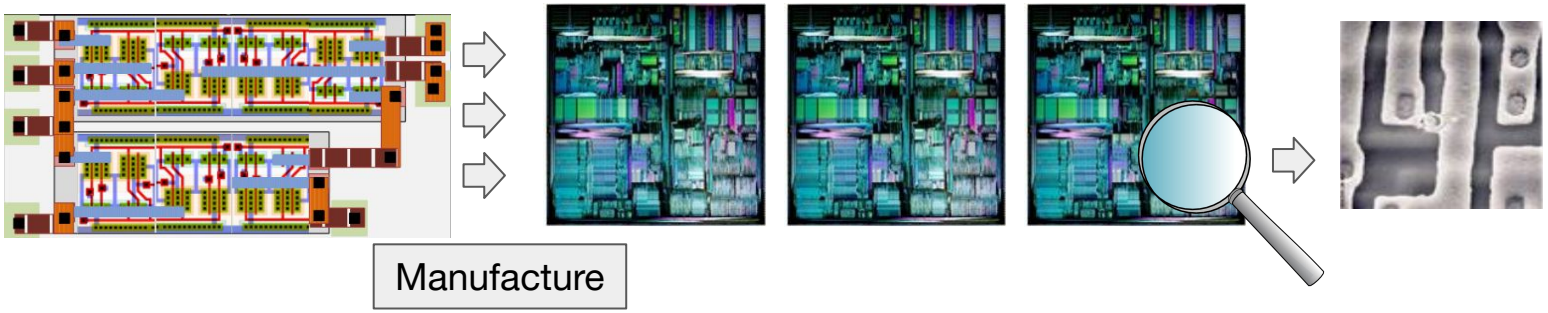
- IBM 4758 -Tamper-proof package; a secure processor + secret key+memory
- Tens of sensors: resistance, temp., voltage
- battery-powered
- ~ $3000 for 99 MHz proc+128MB memory

Solution? “Physically Unclonable Function”

- ✔ maps a challenge to a response
- ✔ is based on a physical system
- ✔ is easy to evaluate (using the physical system)
- ✔ output looks like a random function
- ✔ unpredictable even for attacker w/ physical access

…….. What feature of ICs could we leverage for this?

Manufacturing Variations!

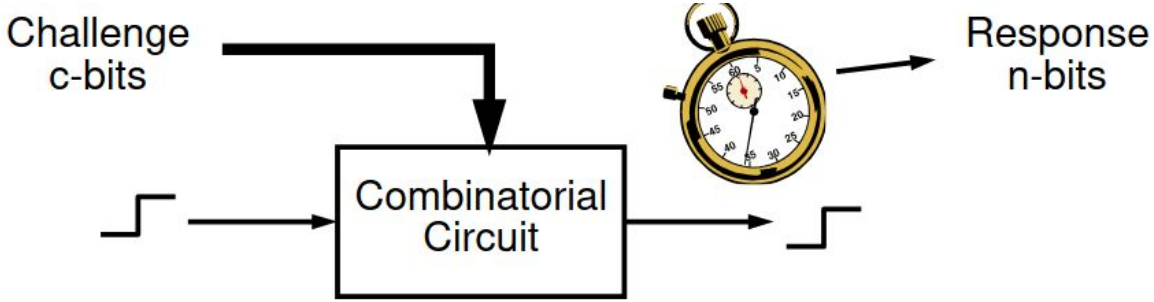


- Every IC, even in same lot, will have inherent variation
  - Due to mask variations, laser variations, temperature…
- These variations cause slight changes in behaviour e.g. delay
  - Magnitude can be up to 5% or more!
- Relative variations increase as device size decrease

PUF from Manufacturing Variations

- Because of random process variations, no two ICs are identical
  - Hard to control or predict
  - Variations increase as fabrication process advances

- Main difference: propagation delays
  - Generate a secret key from unique delay characteristics!



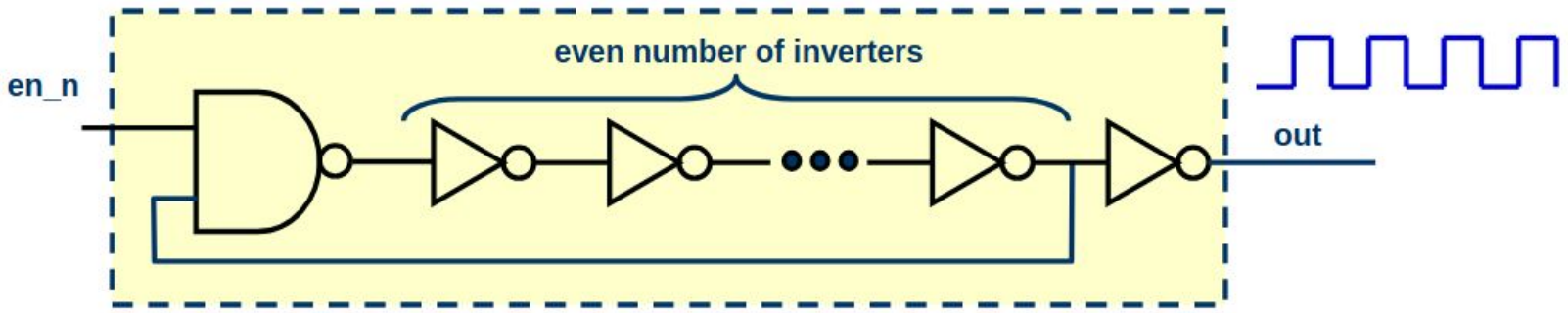
PUFs: the big idea

- Can generate unique/secret key
- Volatile secrets - no need for trusted programming
- Inexpensive - no special fabrication technique
- PUF can enable secure, low-cost auth w/o crypto!
  - Use PUF as a function - challenge ↦ response
  - Only authentic ICs can produce a correct response for a challenge

How to make a PUF?

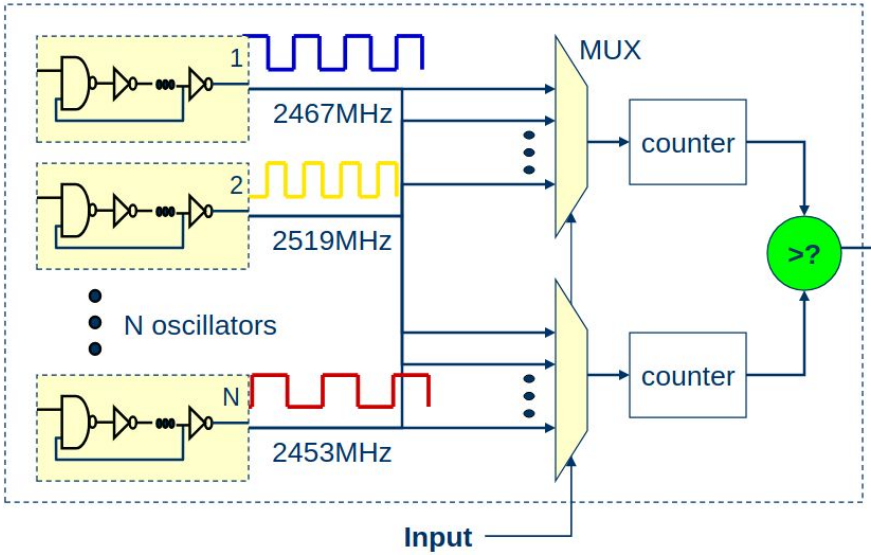
- How to design a PUF circuit?
- How to make a PUF reliable and secure?
- How to use the PUF for key generation and authentication?

Ring Oscillators



- Large combinational loops can form oscillators
- Ring oscillators are a deliberate form of this layout
- They are used to generate clocks or characterize performance
- **Every ring oscillator has a unique frequency even if produced from the same mask**

PUF Using Ring Oscillators



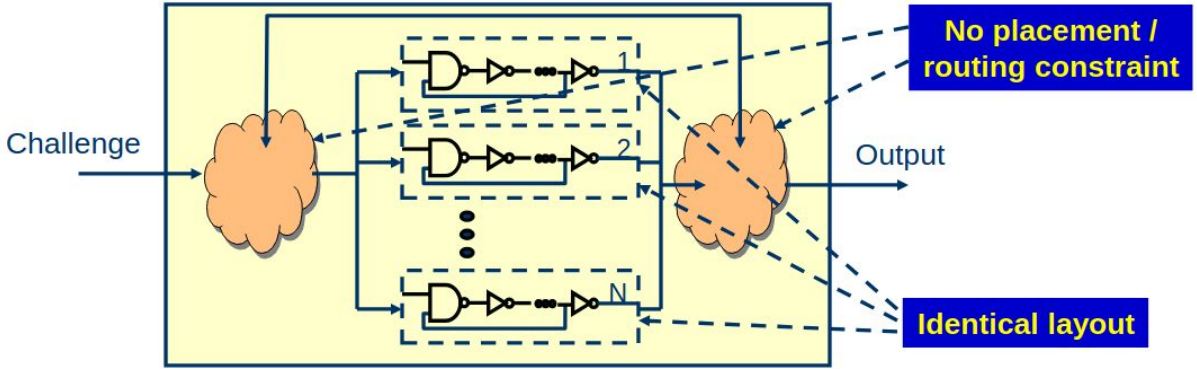
Output (= 0 or 1)

Top	Bot.	Out
1	2	0
1	N	1
2	N	1

Compare frequencies of two oscillators → Faster oscillator is randomly determined by manufacturing variations

Implementation Constraints

- All ring oscillators must be identical
  - Any ring oscillator design will work
- No additional constraints required
  - Everything is standard digital logic
  - No placement/routing constraint outside oscillators
  - Can be implemented even on standard FPGAs (soft PUF)



Entropy: How many bits do you get?

N! possible way N oscillators may be ordered based on their frequencies

– Assume each ordering is equally likely

– 3 oscillators R0, R1, R2 have 6 possible orderings (R0, R1, R2), (R0, R2, R1), (R1, R0, R2), (R1, R2, R0),(R2, R0, R1), and (R2, R1, R0)

Entropy: How many bits do you get?

- N oscillators can produce log2(N!) independent bits
- A reasonable # of oscillators can express keys of long length

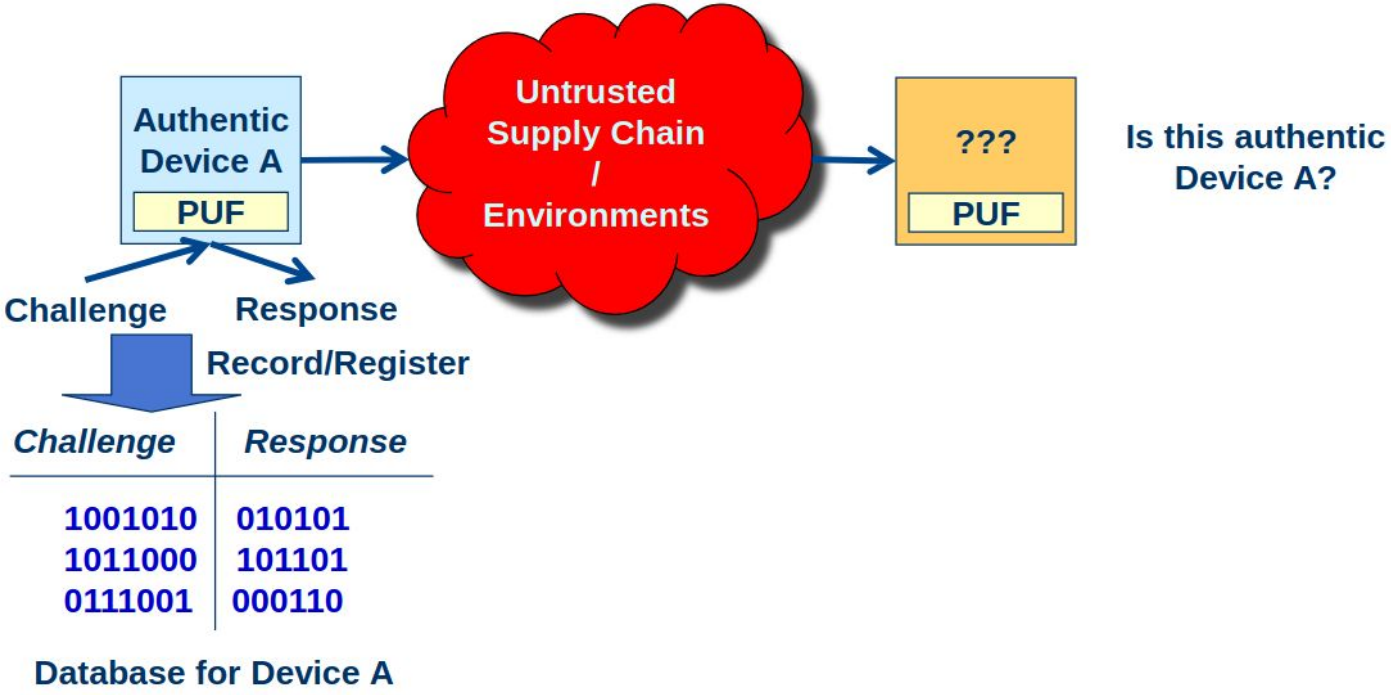
– 35 oscillators produce 133 bits

– 128 oscillators produce 716 bits

– 256 oscillators produce 1687 bits

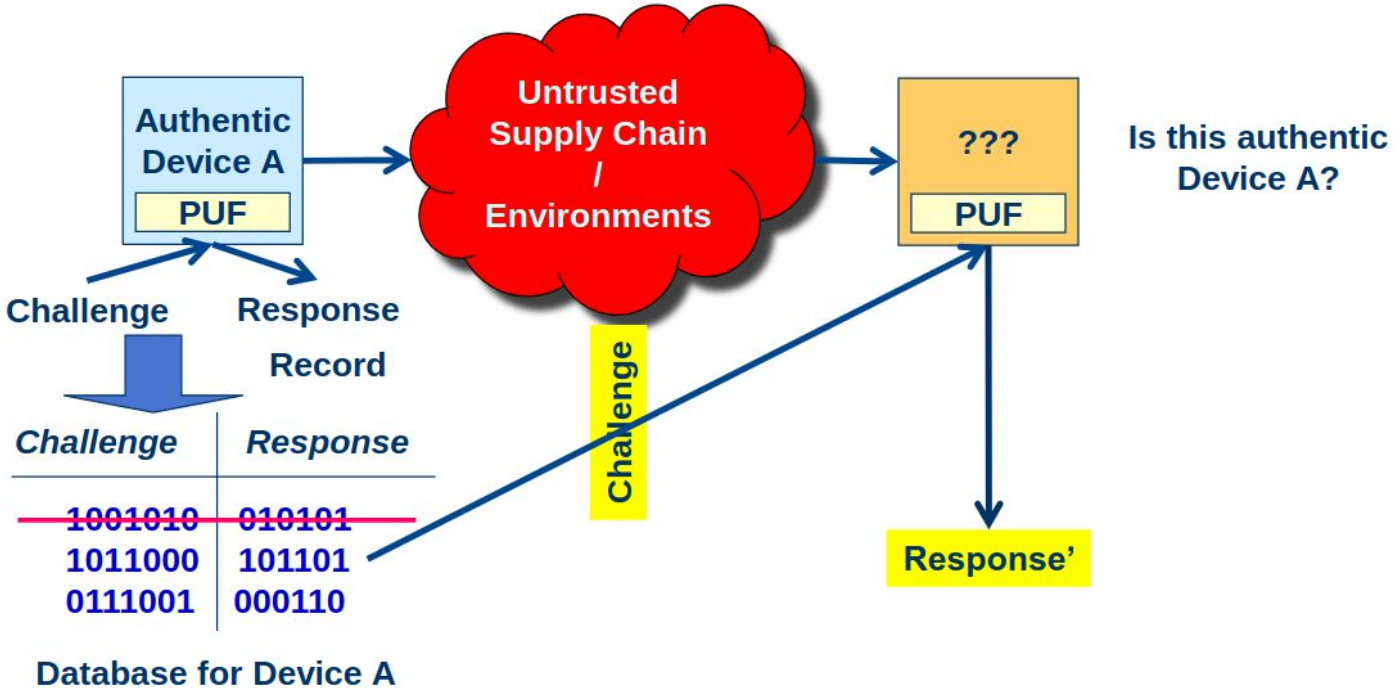
Example: Low-cost Auth

- Protect against IC/FPGA substitution and counterfeits without using cryptographic operations



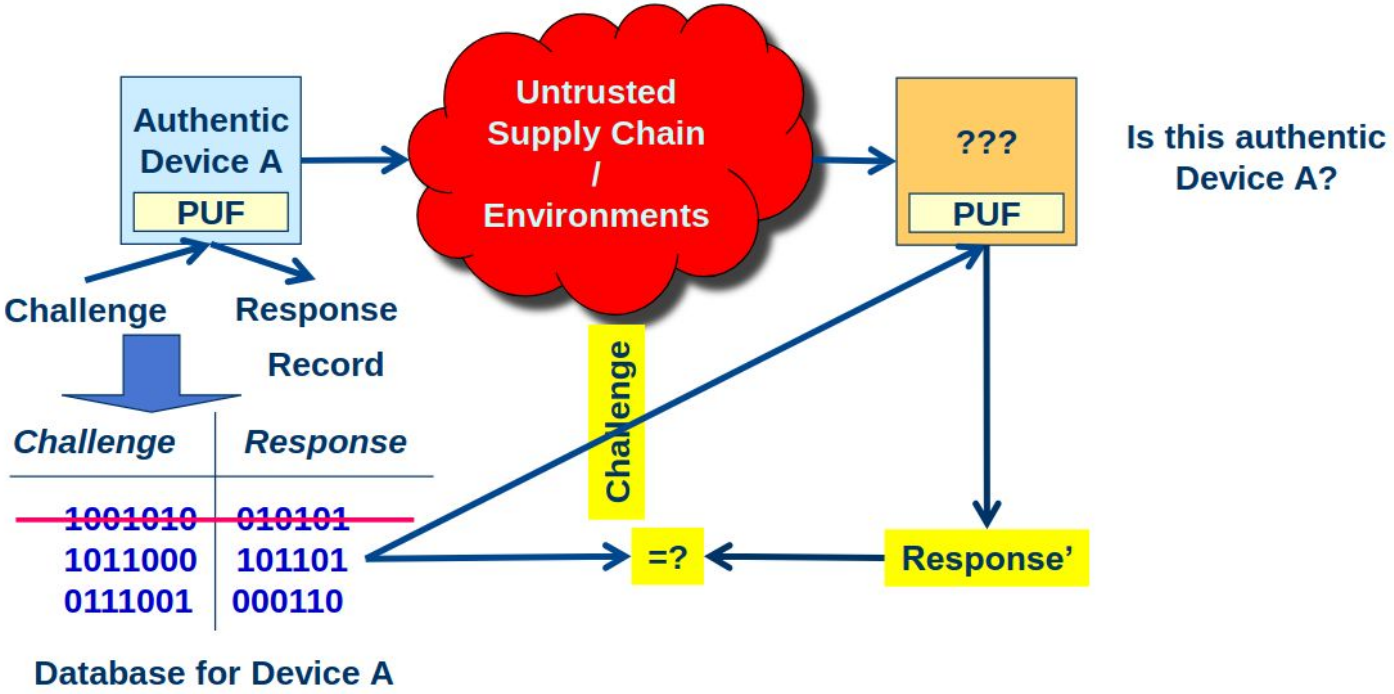
Example: Low-cost Auth

- Protect against IC/FPGA substitution and counterfeits without using cryptographic operations



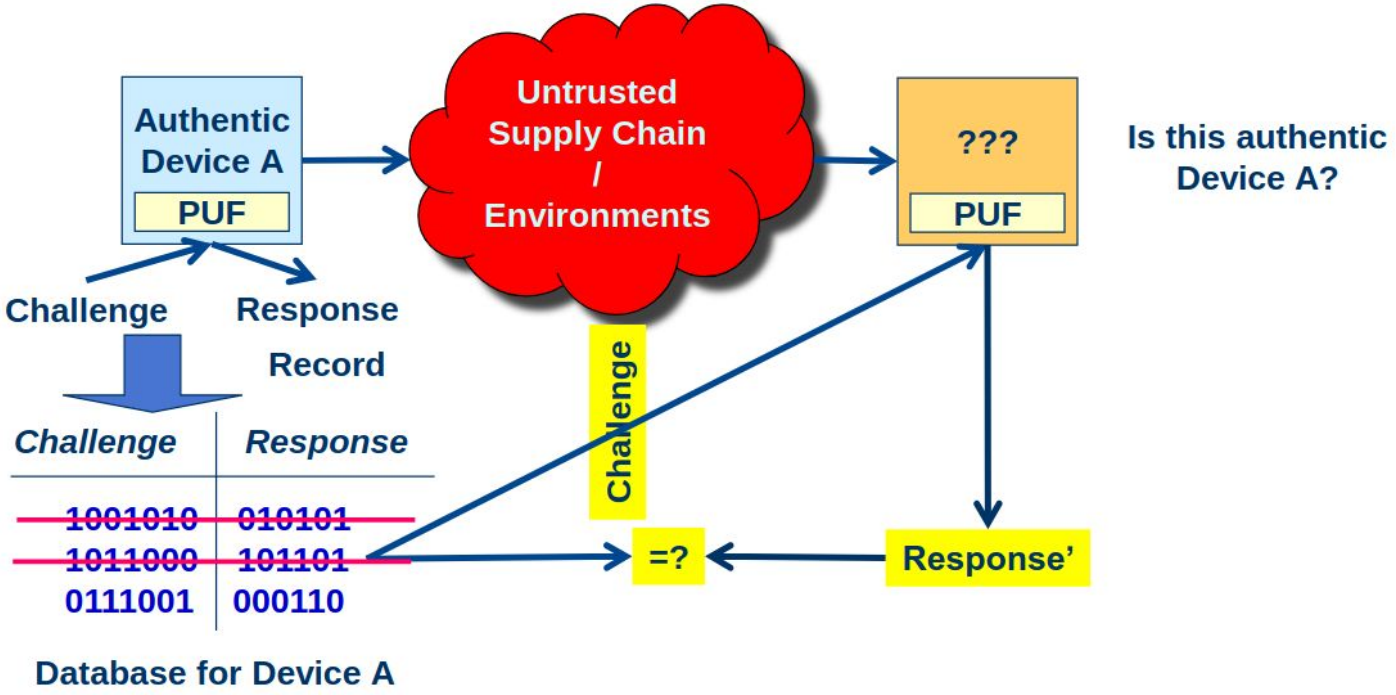
Example: Low-cost Auth

- Protect against IC/FPGA substitution and counterfeits without using cryptographic operations



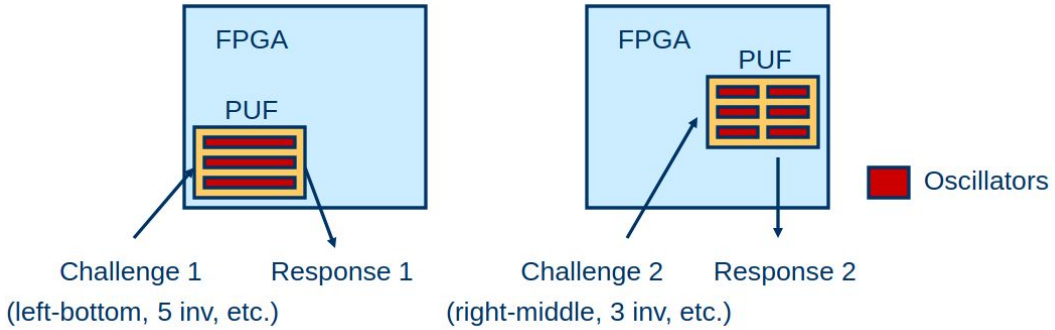
Example: Low-cost Auth

- Protect against IC/FPGA substitution and counterfeits without using cryptographic operations



Challenge-Response Pairs

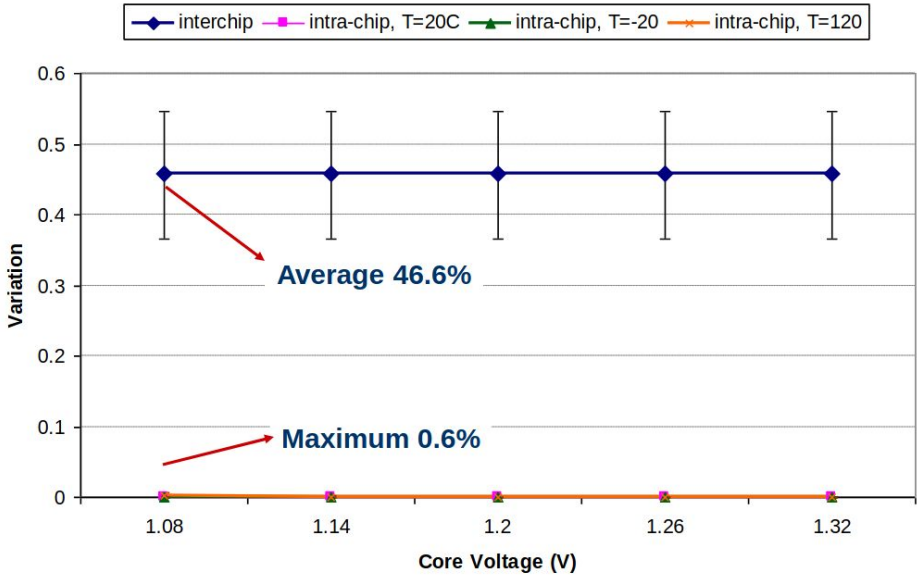
- What if an attacker obtains all responses and stores them into a fake chip with memory?
- There must be LOTS of challenge-response-pairs
  – Use different parts on FPGAs
  – Use configurable delay paths on ASICs



PUF validation

- **Security**: Show that different PUFs (ICs) generate different bits
  - Inter-chip variation: what % of PUF bits differ between PUFs A,B?
  - Ideally, inter-chip variation should be (close to) 50%
- **Reliability**: Show that a PUF (IC) can re-generate consistently
  - Intra-chip variation: how many bits flip when re-generated again from a single PUF
  - Operational environment (voltage, temperature, etc.) can change
  - Ideally, intra-chip variation should be 0%

Example: 1024 oscillators



1024 oscillators at 20C and then again at 120C

Good performance! High interchip and low intra-chip

False Positive and Negative

- False positive: PUF A is authenticated as PUF B

– i.e. PUF A produces same output as PUF B for the same challenge

- False negative: Correct PUF will fail to be authenticated

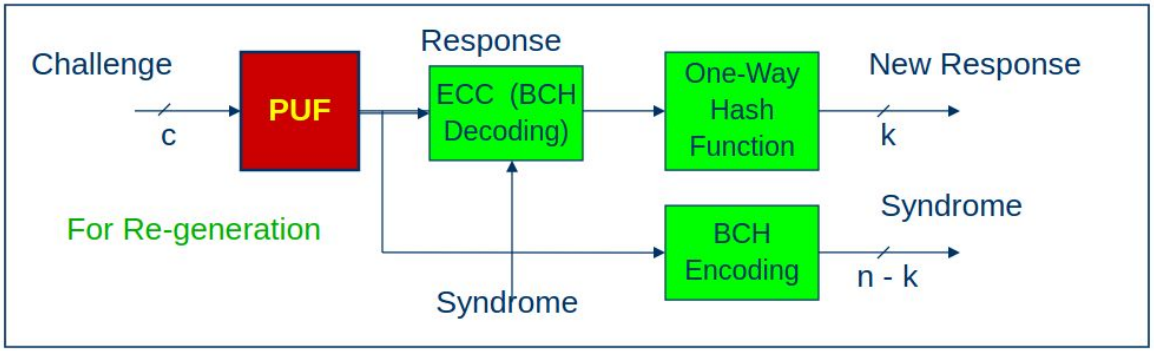
– i.e. the PUF fails to regenerate a consistent output

- If we allow up to 10-out-of-128 bits to be different,

– the False Positive rate is ~2.1x10-21

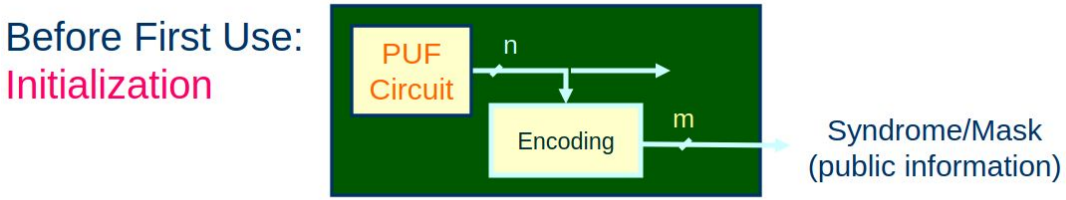
– the False Negative rate is less than 5x10-11

Reliable PUFs



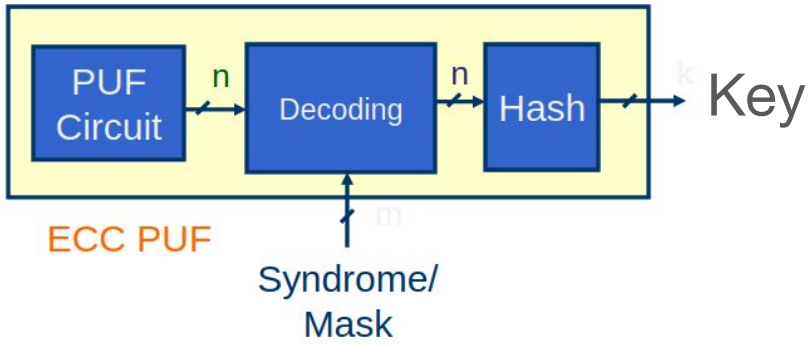
- PUFs can be made secure and reliable by adding extra logic
- Hash function (SHA-x) precludes PUF “model-building” attack
- to obtain PUF output, adversary has to invert a one-way function
- Error Correcting Code (ECC) can eliminate measurement noise without compromising security

Reliable PUF (Initialization)



- To initialize circuit, an error correcting syndrome is generated from reference PUF circuit output
  - Syndrome/error mask is public information
  - Can be stored on-chip, off-chip, or on a remote server
- Example, BCH(127,64,21) code will correct up to 10 errors out of 127 bits
  - Failure probability $< 10^{-10}$

Reliable PUF (+random symmetric key generation)

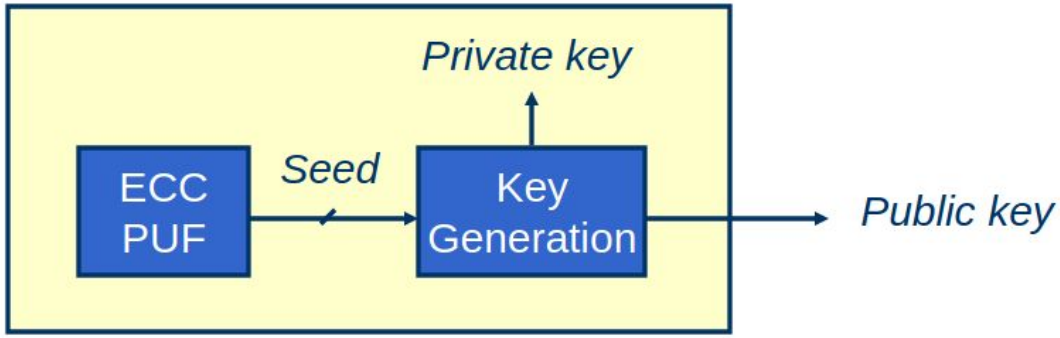


- In the field, the syndrome will be used to re-generate the same PUF reference output from the circuit

– This output is unique and unpredictable for each chip

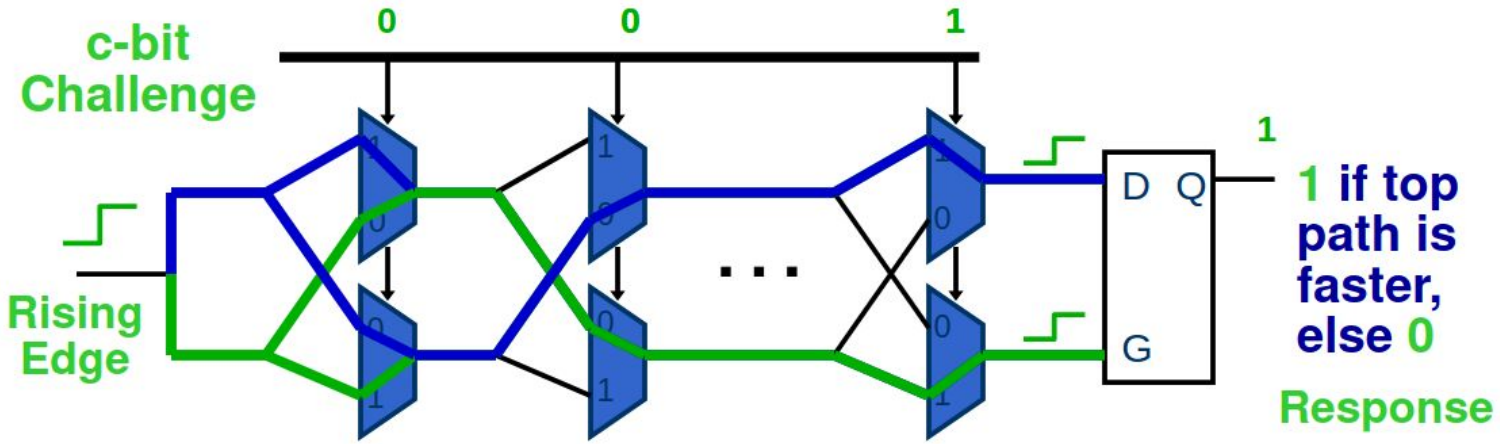
Can be directly used as a symmetric key

ECC PUF for Public/Private Keys



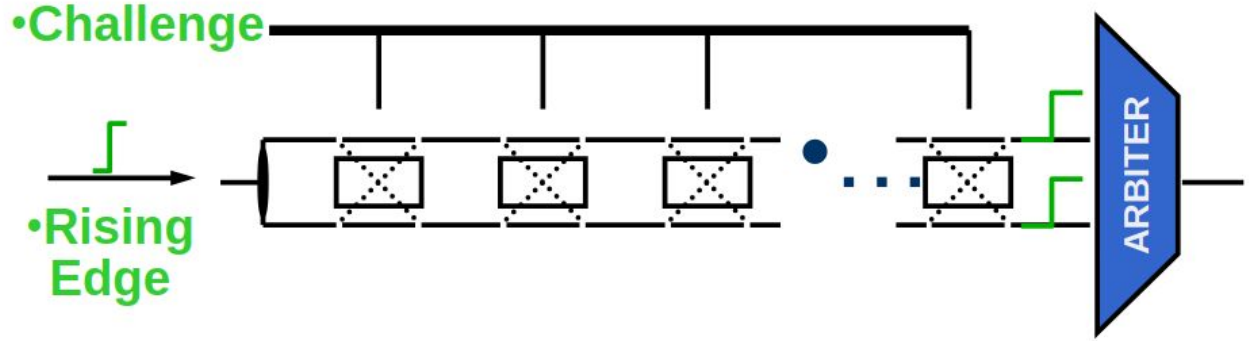
- PUF response is used as a random seed to a private/ public key generation algorithm
  - Secrets are not handled by a manufacturer
- A device generates a key pair on-chip, and outputs a public key
  - The public key can be endorsed at any time

Other types of PUF: Arbiter



- Compare two paths with an identical delay in design
  - Random process variation determines which path is faster
  - An arbiter outputs 1-bit digital response
- Multiple bits can be obtained by either duplicating the circuit or by using different challenges
  - Each challenge selects a unique pair of delay paths

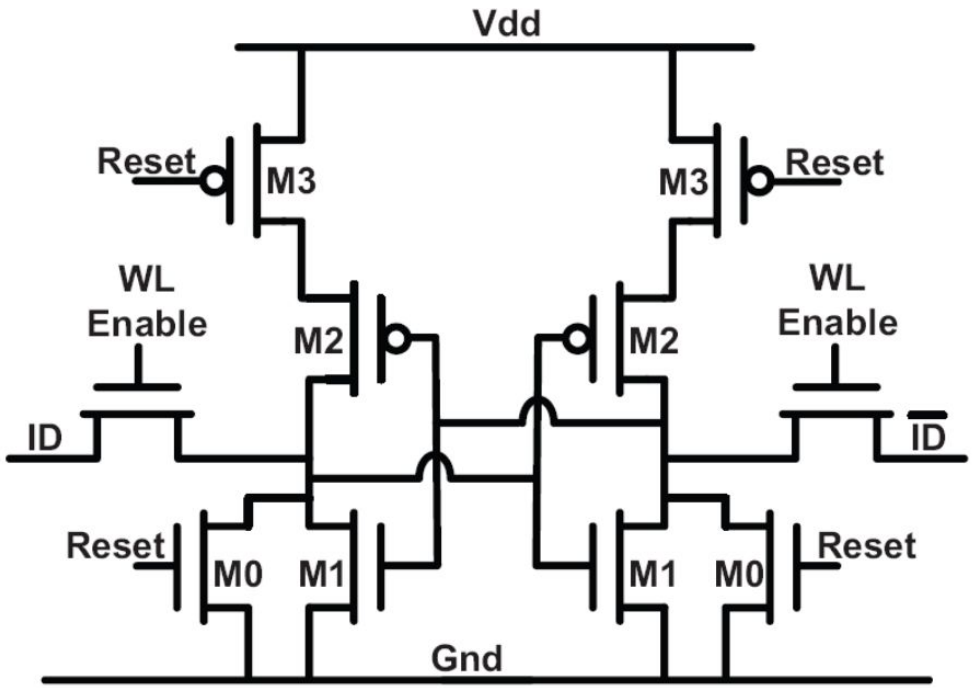
Other types of PUF: Arbiter



Arbiter output is 1 if top path is faster, else 0

- Each challenge creates two paths through the circuit that are excited simultaneously. The digital response is based on a (timing) comparison of the path delays.
- Path delays in an IC are statistically distributed due to random manufacturing variations.

SRAM-based PUF



Random but reliable

One side will energize

Attacking PUFs

- What does it take to find PUF secrets?
- Open the chip, completely characterize PUF delays
  - Invasive attacks are likely to change delays
- Read on-chip register with a digital secret from PUF while the processor is powered up
  - More difficult proposition than reading non-volatile memory
- Still, this only reveals ONE secret from PUF
  - Use two keys with only one key present on-chip at any time
- PUF is a cheap way of generating more secure secrets

Biggest challenge

- PUFs inherently unstable
- This instability can be leveraged to cause auth failures
  - E.g. too many bit flips for the ECC
  - Inauthentic chip reads as authentic, authentic chips reads inauthentic
- More reading:
  - Temperature-Aware Cooperative (TAC) ring oscillators

Takeaways

- PUFs leverage process variations as a feature
- PUFs generate unique/unpredictable secrets for each IC
  - Highly secure/volatile keys
  - Inexpensive: standard digital logic
- PUFs can be used to:
  - Auth ICs without cryptographic operations
  - Generate symmetric and asymmetric keys for crypto operations

- PUFs are used in FPGAs, ASICs, even passive RFIDs!

Assessments - Week 9

Lab5 - PUFs

Lab5 - PUF

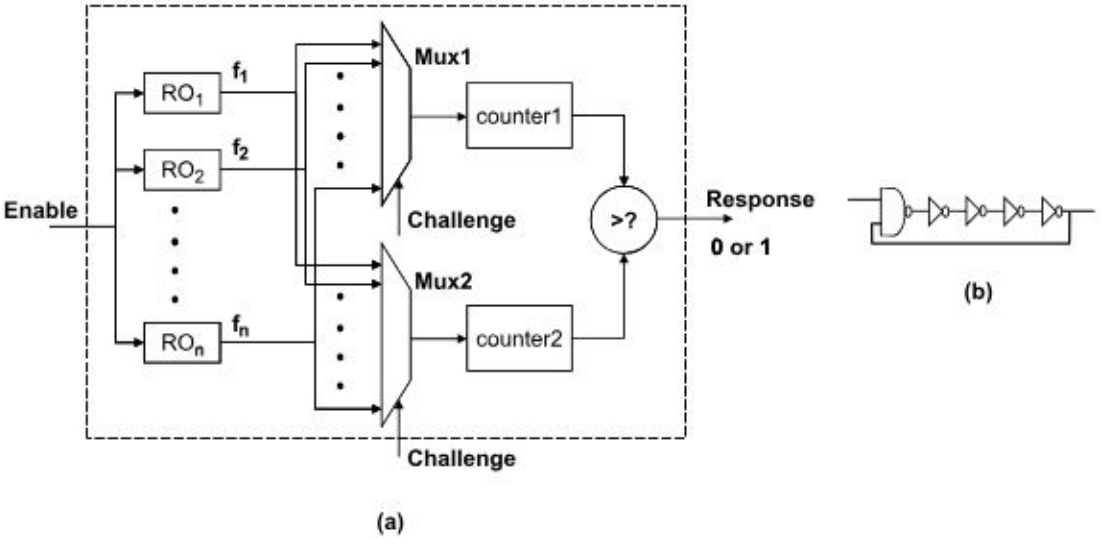


Fig. 1. (a) Ring oscillator PUF scheme. (b) A basic five-stage ring oscillator loop.

- You're going to build and characterize some ring oscillators/PUFs!
- Measure your hackster against other(s) in the class.

More details in the lab instructions!

Due Week 10 Friday