Hardware Security

Chester Rebeiro
IIT Madras
Physically Unclonable Functions

Physical Unclonable Functions and Applications: A Tutorial
Edge Devices

1000s of them expected to be deployed

Low power (solar or battery powered)
Small footprint
Connected to sensors and actuators

Expected to operate 24 x 7 almost unmanned

24x7 these devices will be continuously pumping data into the system, which may influence the way cities operate

Will affect us in multiple ways, and we may not even know that they exist.
Authenticating Edge Devices

- Stored keys
  - EEPROM manufacture is an overhead
  - Public key cryptography is heavy
  - Can be easily copied / cloned

Public keys stored in server

Encryption done in edge device

Private keys
Physically Unclonable Functions

- No stored keys
- No public key cryptography
- Cannot be cloned / copied
- Uses nano-scale variations in manufacture. No two devices are exactly identical

Digital Fingerprints

Public keys stored in server

Encryption done in edge device

Challenge / response
A function whose output depends on the input as well as the device executing it.
What is Expected of a PUF?
(Inter and Intra Differences)

(Reliable)
Same Challenge to Same PUF
Difference between responses must be small on expectation

Irrespective of temperature, noise, aging, etc.

(Unique)
Same Challenge to different PUF
Difference between responses must be large on expectation

Significant variation due to manufacture
What is Expected of a PUF? (Unpredictability)

Difficult to predict the output of a PUF to a randomly chosen challenge when one does not have access to the device.
Intrinsic PUFs

• Completely within the chip
  – PUF
  – Measurement circuit
  – Post-processing
    • No fancy processing steps!
  – eg. Most Silicon based PUFs
Silicon PUFs

eg. Ring Oscillator PUF

Ring Oscillator with odd number of gates

\[ f = \frac{1}{2nt} \]

- \( f \) Frequency of ring oscillator
- \( n \) Number of stages
- \( t \) Delay of each stage

Frequency affected by process variation.
Why variation occurs?

**MOS Transistor**

- Source
- Oxide
- Gate
- Drain
- Body

**CMOS Inverter**

Delay depends on capacitance

When gate voltage is less than threshold no current flows.

When gate voltage is greater than threshold current flows from source to drain.

Threshold voltage is a function of doping concentration, oxide thickness.

Process Variations
- Oxide thickness
- Doping concentration
- Capacitance
Silicon PUFs

eg. Ring Oscillator PUF

N bit challenge

\[
\text{response} = \begin{cases} 
1 & f_A > f_B \\
0 & f_A \leq f_B 
\end{cases}
\]
Results of a RO PUF

15 Xilinx, Virtex 4 FPGAs;
1024 ROs in each FPGA;
Each RO had 5 inverter stages and 1 AND gate

Inter Chip Variations
(Uniqueness measurement)

When 128 bits are produced,
Avg 59.1 bits out of 128 bits different

Physical Unclonable Functions for Device Authentication and Secret Key Generation
Results of a RO PUF

15 Xilinx, Virtex 4 FPGAs;
1024 ROs in each FPGA;
Each RO had 5 inverter stages and 1 AND gate

Intra Chip Variations
(Reproducability measurement)

0.61 bits on average out of 128 bits differ

Physical Unclonable Functions for Device Authentication and Secret Key Generation
Ideally delay difference between Red and Blue lines should be 0 if they are symmetrically laid out. In practice variation in manufacturing process will introduce random delays between the two paths.
If the signal at D reaches first then Q will be set to 1
If the signal at clk reaches first then Q will be set to 0
Arbiter PUF

13.56MHz Chip
For ISO 14443 A spec.
Results for RO PUF

Design and Implementation of PUF-Based “Unclonable” RFID ICs for Anti-Counterfeiting and Security Applications
IEEE Int.Conf. on RFID, 2008, S. Devdas et. Al.
Comparing RO and Arbiter PUF

Number of Challenge : \[ N \]
Response Pairs : \[ \frac{N}{2} \]

#CRPs linearly related to the number of components

WEAK PUF

Number of Challenge : \[ 2^N \]
Response Pairs : \[ 2^N \]

#CRPs exponentially related to the number of components

STRONG PUF
### Weak PUF vs Strong PUF

<table>
<thead>
<tr>
<th>Weak PUF</th>
<th>Strong PUF</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Very Good Inter and Intra differences</td>
<td>• Huge number of Challenge Response Pairs (CRPs)</td>
</tr>
<tr>
<td>• Comparatively few number of Challenge Response Pairs (CRPs)</td>
<td>• It is assumed that an attacker cannot Enumerate all CRPs within a fixed time interval. Therefore CRPs can be made public</td>
</tr>
<tr>
<td>• CRPs must be kept secret, because an attacker may be able to enumerate all possible CRPs</td>
<td>• Formally, an adversary given a poly-sized sample of adaptively chosen CRPs cannot predict the Response to a new randomly chosen challenge.</td>
</tr>
<tr>
<td>• Weak PUFs useful for creating cryptographic keys</td>
<td>• Does not require any cryptographic scheme, since CRPs can be public.</td>
</tr>
<tr>
<td>• Typically used along with a cryptographic scheme (like encryption / HMAC etc) to hide the CRP (since the CRPs must be kept secret)</td>
<td></td>
</tr>
</tbody>
</table>
PUF Based Authentication (with Strong PUF)

**Bootstrapping**: At manufacture, server builds a database of CRPs for each device.

At deployment, server picks a random challenge from the database, queries the device and validates the response.
Man in the middle may be able to build a database of CRPs. To prevent this, CRPs are not used more than once.
PUF Based Authentication

CRP Tables

Each device would require its own CRP table and securely stored in a trusted server. Tables must be large enough to cater to the entire lifetime of the device or need to be recharged periodically (scalability issues).
PUF based Authentication
(Alleviating CRP Problem)

Secret Model of PUF

**Gate Delays of PUF components**

**Bootstrapping:** At manufacture, server builds a database of gate delays of each component in the PUF.

At deployment, server picks a random challenge, constructs its expected response from secret model, queries the device and validates the response.

**Still Requires Secure Bootstrapping and Secure Storage**
PPUF: Public Model PUF

- Gate Delays of PUF Components (Public)
- Trusted server (PKI)

Bootstrapping: Download the public model of PUF from the trusted server.

At deployment, server picks a random challenge constructs expected response from public model, queries the device and validates the response. If time for response is less than a threshold accept response else rejects.

Assumption: A device takes much less time to compute a PUF response than an attacker who models the PUF.

\[ T < T_0 \]
PUF based Authentication
(Alleviating CRP Problem)

Homomorphic Encryption
Conclusions

• Different types of PUFs being explored
  – Analog PUFs, Sensor PUFs etc.

• CRP issue still a big problem

• Several attacks feasible on PUFs.
  – Model building attacks (SVMs)
  – Tampering with PUF computation (eg. Forcing a sine-wave on the ground plane, can alter the results of the PUF)

• PUFs are a very promising way for lightweight authentication of edge devices.
Hardware Trojans
Cyber-attack concerns raised over Boeing 787 chip's 'back door'

Researchers claim chip used in military systems and civilian aircraft has built-in function that could let in hackers

Western spooks banned Lenovo PCs after finding back doors

Report suggests 'Five Eyes' alliance won't work with Chinese PCs

By Phil Muncaster 29 Jul 2013 at 03:45

NSA Subverts Most Encryption, Works With Tech Organizations For Back-Door Access, Report Says

Posted Sep 5, 2013 by Gregory Ferenstein (ferenstein)

NSA’s Own Hardware Backdoors May Still Be a “Problem from Hell”

Revelations that the NSA has compromised hardware for surveillance highlights the vulnerability of computer systems to such attacks.

https://www.theguardian.com/technology/2012/may/29/cyber-attack-concerns-boeing-chip
https://techcrunch.com/2013/09/05/nsa-subverts-most-encryption-works-with-tech-companies-for-back-door-access-report-says/
https://www.theregister.co.uk/2013/07/29/lenovo_accused_backdoors_intel_ban/
IC Life Cycle (Vulnerable Steps)

Malware in Third Party IPs

• Third party IPs
  – Can they be trusted?
  – Will they contain malicious backdoors

• Developers don’t / can’t search 1000s of lines of code looking out for trojans.
FANCI: Identification of Stealthy Malicious Logic

• FANCI: evaluate hardware designs automatically to determine if there is any possible backdoors hidden

• The goal is to point out to testers of possible trojan locations in a huge piece of code

(some of the following slides are borrowed from Waksman’s CCS talk)
Hardware Trojan Structure

Trojan can be inserted anywhere in during the manufacturing process (eg. In third party IP cores purchased, by fabrication plant, etc.)

**Trigger Circuit:**
Based on a seldom occurring event. For example,
- when address on address bus is 0xdeadbeef.
- A particularly rare packet arrives on network
- Some time has elapsed

**Payload:**
Do something nefarious:
- Make a page in memory (un)privileged
- Leak information to the outside world through network, covert channels, etc
- Cause the system to fail
Trojan=Trigger+Payload

Ex: AES Key Stealing  Ciphertext  Key Exfiltration

Instead of generating Ciphertext...

0xba5eba11

AES-P

\* n = 127 for AES128-P core.
\* n = 31 for AES32-P core.
Trojan=Trigger+Payload

Ex: AES Key Stealing  Ciphertext  Key Exfiltration

...a backdoor can give access to the key!

0xba5eba11
Backdoors are Stealthy

• **Small**
  – Typically a few lines of code / area

• **Stealth**
  – Cannot be detected by regular testing methodologies (rare triggers)
  – Passive when not triggered
Unfortunately...

With so much of code it is highly likely that stealthy portions of the code are missed or not tested properly.

FANCI: will detect these stealthy circuits. These parts are most likely to have Trojans.
The aim is to have no false negatives.
A few false positives are acceptable.
Control Values

By how much does an input influence the output O?

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>O</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Control Values

By how much does an input influence the output 0?

A: has a control of 0.5 on the output

(A matters in this function)
Control Values

By how much does an input influence the output $O$?

A : has a control of 0 on the output

(A does not matter in this function)
(A is called unaffected)

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
A31 has a control value $1/2^{32}$

Easier to hide a trojan when larger input sets are considered

A low chance of affecting the output lends itself to stealthiness → easier to hide a malicious code

<table>
<thead>
<tr>
<th>A31</th>
<th>A30</th>
<th>A2</th>
<th>A1</th>
<th>A0</th>
<th>trigger</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>...</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>...</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>...</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>...</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>:</td>
<td>:</td>
<td>:</td>
<td>:</td>
<td>:</td>
<td>:</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>:</td>
<td>:</td>
<td>:</td>
<td>:</td>
<td>:</td>
<td>:</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
An Example of a Mux

\(<A, B, C, D, S1, S2> = <0.25, 0.25, 0.25, 0.25, 0.5, 0.5>\)

No trojan present here (intuitively):

* All mux inputs have a control value around mid range (not too close to 0)
An Example of a Malicious Mux

66 extra select lines which are only modify M when they are set to a particular value

The control values E and S3 to S66 are suspicious because they rarely influence the value of M.

Perfect for disguising malicious backdoors

Just searching for MIN values is often not enough. Better metrics are needed.
Computing Stealth from Control

We use three different heuristics for evaluation. Mean, Median and Triviality.

Mean(M) = (2.0 / 6) = 0.33
Median(M) = 0.25
Triviality(M) = 0.50

-The Median in the context of backdoor triggers is often close to zero when low or unaffected wires are present.
-The Mean is sensitive to outliers. If there are few dependencies, and one of them is unaffected, it is likely to get noticed, when compared to the control value.
-Triviality is a weighted average of the values in the vector. Weighted by how often they are the only value affecting the output. If it is 0 or 1 it is trivial.
Computing Stealth from Control

Mean(M) = (2.0 / 71) = 0.03
Median(M) = $2^{-63}$
Triviality(M) = 0.50
FANCI: The Complete Algorithm

Algorithm 1 Flag Suspicious Wires in a Design

1: for all modules $m$ do
2:   for all gates $g$ in $m$ do
3:     for all output wires $w$ of $g$ do
4:       $T \leftarrow \text{TruthTable}($FanInTree$(w))$
5:     $V \leftarrow \text{Empty vector of control values}$
6:     for all columns $c$ in $T$ do
7:       Compute control of $c$ (Section 3.2)
8:       Add control($c$) to vector $V$
9:     end for
10:  Compute heuristics for $V$ (Section 3.3)
11:  Denote $w$ as suspicious or not suspicious
12: end for
13: end for
14: end for
IC Life Cycle (The Fab)

Third-party

Detecting Trojans in ICs

• Optical Inspection based techniques
  Scanning Optical Microscopy (SOM),
  Scanning Electron Microscopy (SEM),
  and pico-second imaging circuit analysis (PICA)
  – Drawbacks: Cost and Time!

• Testing techniques
  – Not a very powerful technique

• Side channel based techniques
  – Non intrusive technique
  – Compare side-channels with a golden model

A Survey on Hardware Trojan Detection Techniques
http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=7169073
Side Channel Based Trojan Detection

Lightweight PRESENT Implementation

Power Traces

Hardware trojan design and detection: a practical evaluation

https://dl.acm.org/citation.cfm?id=2527318
Side Channel Based Trojan Detection (IC with Trojan)
Difference of Distributions
Hardware Trojan Prevention
(If you can’t detect then prevent)

Backdoor = Trigger + Payload

Silencing Hardware Backdoors
Slides taken from Adam Waksman’s Oakland talk
Hardware Trojan Prevention

Ensure that a hardware Trojan is never delivered the correct Trigger
Example (A 5 stage processor)

- A design is a connected set of modules
  - Modules connect to each other through interfaces

- In the picture above, each box is a module
Example (A 5 stage processor)

- A design is a connected set of modules
  - Modules connect to each other through interfaces

- In the picture above, each box is a module
Types of Trojans
Ticking Timebomb

- After a fixed time, functionality changes.
Ticking Timebomb
Cheat Codes

• A special value turns on malicious functionality
  • Example: 0xcafebeef
Cheat Codes

- Example: 0xcaffebef
Sequence Cheat Codes

- A set of bits, events, or signals cause malicious functionality to turn on
  - Example: c, a, f, e, b, e, e, f
Hardware Trojan Silencing (with Obfuscation)
Silencing Ticking Timebombs

- Power Resets: flush pipeline, write current IP and registers to memory, save branch history targets

- Power to modules is reset periodically
  - Time period = N - K cycles
  - N = Validation epoch
  - K = Time to restart module operation

- Forward progress guarantee
  - Architectural state must be saved and restored
  - Microarchitectural state can be discarded (low cost)
    - e.g., branch predictors, pipeline state etc.,
Silencing Ticking Timebombs

• Can trigger be stored to architectural state and restored later
  – No. Unit validation tests prevent this
  – Reason for trusting validation epoch
    Large validation teams
    Organized hierarchically

• Can triggers be stored in non-volatile state internal to the unit?
  – Eg. Malware configures a hidden non-volatile memory

• Unmaskable Interrupts?
  – Use a FIFO to store unmaskable interrupts

• Performance Counters are hidden time bombs
Data Obfuscation

Homomorphic Encryption (Gentry 2009)
Ideal solution
But practical hurdles
Data Obfuscation

Non-Computational Case

Router, Interconnect, Memory, Cache, Comparator, Buffer, Register, Table, etc.
Data Obfuscation

Non-Computational Case
Router, Interconnect, Memory, Cache, Comparator, Buffer, Register, Table, etc.

Memory Controller

Store Data 5 to Address 7
Data Obfuscation (Computational Case)

Computational Case

ALUs, FGUs, decoders, custom logic, etc.
Sequence Breaking
(Reordering)

Ensure functionality is maintained
Sequence Breaking (Inserting events)

Insert arbitrary events when reordering is difficult
Expensive:
Non-recurring: design; verification costs due to duplication
Recurring: Power and energy costs