Why We Should Be Worried about Hardware Trojans

The Summer Research Institute 2018
EPFL, June 18, 2018

Christof Paar
Ruhr Universität Bochum & University of Massachusetts Amherst
Acknowledgement

- Georg Becker
- Pawel Swierczynski
- Marc Fyrbiak
Agenda

- Introduction to Hardware Trojans
- Sub-Transistor ASIC Trojans
- FPGA Trojan
- Key extraction attack
- Auxiliary Stuff
Agenda

- Introduction to Hardware Trojans
- Sub-Transistor ASIC Trojans
- FPGA Trojan
- Key extraction attack
- Auxiliary Stuff
Hardware Trojans

Malicious change or addition to an IC that adds or remove functionality, or reduces reliability

Many rather unpleasant “applications”
Hardware Trojans & the Scientific Community

Publications w/ "Hardware Trojans" or "malicious Hardware" (Google Scholar, Oct 2017) only

Defense Science Board Task Force
On
HIGH PERFORMANCE MICROCHIP SUPPLY

Department of Defense
United States of America

February 2005

Office of the Under Secretary of Defense
For Acquisition, Technology, and Logistics
Washington, D.C. 20301-3140
Trojan Injection & Adversaries Scenarios

DoD scenario 2005

- **Manufacturing**
  Malicious factory, esp. off-shore (foreign Government)

- **Design Manipulation**
  - 3rd party IP-cores
  - malicious employee

not-so-unlikely 2013

- **During shipment**
- **Built-in**
  backdoors etc.


NSA’s interdiction
Where are we with “real” HW Trojans?

- No true hardware Trojan observed in the wild
- All examples from academia
- Vast majority of publications focus on detection
Agenda

- Introduction to Hardware Trojans
- Sub-Transistor ASIC Trojans
- FPGA Trojan
- Key extraction attack
- Auxiliary Stuff
Our Thoughts

1. *Designing* Trojan could be fun too
2. Especially those that go *undetected*
Simple Example: Inverter Trojan

Let’s modify an inverter so that it always outputs “1” (VDD) without visible changes.

<table>
<thead>
<tr>
<th>A</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

![Diagram of inverter with modifications]
PMOS Transistor Trojan

Unmodified PMOS transistor

Trojan trans. w/ constant VDD output
Q1: Can the manipulation be detected?
Q2: How to build a useful Trojan from here?
Detection: layout view of Trojan inverter

Which one has the Trojan?

Original Inverter

“Always One” Trojan

Unchanged:
- All metal layers
- Polysilicon layer
- Active area
- Wells

⇒ Dopant changes (very ?) difficult to detect using optical inspection!
“Small” remaining question

- Unfortunately, we merely introduce a stuck-at fault ...
- ... functional testing (after manufacturing) will detect fault right away

Q2: Can we build a meaningful Trojan using dopant modifications that passes functional testing?
A Real-World True Random Number Generator

Disclaimer: Attacks works against most modern TRNGs

dopant Trojan

... random numbers generate cryptographic keys for

- secure web browsing
- email encryption
- document certification
- ...

Secure Sign In
- Online ID:
- Save this Online ID
- I forgot my Online ID
- Check Your SiteKey
2 Modules form Random Number Generator

- entropy source
- digital post processing

128 → Crypto Key

011001011110 ...
Inside the Random Number Generator

- 011001011110 ...
- 256 random bits
- testing all keys: lifetime of the universe

State register \( k \):

\[
\begin{array}{cccccccccc}
0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & \ldots & 0
\end{array}
\]

State register \( c \):

\[
\begin{array}{cccccccccc}
1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & \ldots & 1
\end{array}
\]

AES

Crypto Key

- 1,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000 possible crypto keys
Trojan Random Number Generator

224 Trojan bits (fixed by attacker!)

\[ \begin{array}{cccccccc}
0 & 1 & 1 & 0 & 1 & 1 & 0 & 1 \\
\end{array} \ldots \begin{array}{c}
1
\end{array} \]

\[ \begin{array}{cccccccc}
c_1 & c_2 & \ldots & c_{32} & 0 & 0 & 1 & \ldots & 0
\end{array} \]

- \[ 1,000,000,000,000,000,000,000,000,000,000,000,000,000 \] possible crypto keys

only 32 random bits

Testing all keys: few seconds

\[ 128 \rightarrow \text{AES} \rightarrow 128 \rightarrow \text{Crypto key} \]

... but circuit would still be tested as “faulty” during manufacturing...
Built-in self test prevents detection of fault

known input

256 bit state
Digital Post Processing (AES)

512 bits

CRC Checksum

32 bits

Reference Checksum

Due to clever choosing of the Trojan bits

TROJAN

known input

Digital Post Processing (AES)

512 bits

CRC Checksum

32 bits

Reference Checksum

≠

=
Conclusion

- Meaningful hardware Trojans are possible without extra logic
- Many detection techniques don’t guarantee a Trojan free design!
- Built-in self tests can be dangerous
- More details:
  Becker, Regazzoni, P, Burleson, *Stealthy Dopant-Level Hardware Trojans*. CHES 2013

... but the scientific community functions as it is supposed to do:

- Trojan detection is possible w/ scanning electron microscope
  Sugawara et al., *Reversing Stealthy Dopant-Level Circuits*. CHES 2014
Agenda

- Introduction to Hardware Trojans
- Sub-Transistor ASIC Trojans
- FPGA Trojan
- Key extraction attack
- Auxiliary Stuff
FPGAs = Reconfigurable Hardware
... are widely used
Can we build hardware Trojans by manipulating the bitstream?
Principle of FPGA-based Trojans

Manipulate Bits

Source Graphics: SimpleIcon, Xilinx
The Mechanics of FPGAs

Two challenges
1. find AES in unknown design
2. meaningful manipulation

FPGA fabric

$10^3 \ldots 10^6$ logic cells

bitstream is complex and proprietary

100101010101010101010100
0011101001011011100000
0001010111010100110011
101011001100101011111
Finding AES:
Luckily, crypto has very specific components

- S-boxes are realized as 6x1 look-up tables (LUTs)
- LUT locations can be „easily“ found in bitstream
- S-box contents is very specific (luckily)
AES detection in practice

8 different real-world AES implementations

<table>
<thead>
<tr>
<th>Impl.</th>
<th>Architecture</th>
<th>AES</th>
<th>LUTs with S-box logic</th>
<th>S-boxes in memory</th>
<th>Detection</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td>Round-based</td>
<td>128</td>
<td>(16+4) \cdot 32 = 640</td>
<td>no</td>
<td>100 %</td>
</tr>
<tr>
<td>#2</td>
<td>Round</td>
<td>128</td>
<td>0</td>
<td>yes</td>
<td>100 %</td>
</tr>
<tr>
<td>#3</td>
<td>(\frac{1}{4}) Round</td>
<td>192</td>
<td>0</td>
<td>yes</td>
<td>100 %</td>
</tr>
<tr>
<td>#4</td>
<td>(\frac{1}{4}) Round</td>
<td>256</td>
<td>0</td>
<td>yes</td>
<td>100 %</td>
</tr>
<tr>
<td>#5</td>
<td>Round-based</td>
<td>128</td>
<td>(0+4) \cdot 32 = 128</td>
<td>yes</td>
<td>100 %</td>
</tr>
<tr>
<td>#6</td>
<td>Round-based</td>
<td>128</td>
<td>0</td>
<td>yes</td>
<td>100 %</td>
</tr>
<tr>
<td>#7</td>
<td>Round-based</td>
<td>128</td>
<td>0</td>
<td>yes</td>
<td>100 %</td>
</tr>
<tr>
<td>#8</td>
<td>Round-based</td>
<td>128</td>
<td>(16+4) \cdot 32 = 640</td>
<td>no</td>
<td>100 %</td>
</tr>
</tbody>
</table>

TABLE IV: Overview of evaluated AES implementations
Algorithm substitution attack and its implications

1. Inject weak S-boxes in bitstream

2. Trojan AES is configured

"Useful" attacks are still possible!

1. Storage encryption – Plaintext recovery
   - Attacker can recover plaintext without access to $k$

2. Temporary device access – Key extraction
   - switch S-box and recover $k$ from $CT$
   - configure original S-box
Conclusion

- New attack vector against FPGAs!
- Reconfigurability allows “hardware” Trojans designed in the lab
- Bitstream protection is crucial!
  (but not easy, cf. our work at CCS 2011 & FPGA 2013)
- Details at:
Agenda

- Introduction to Hardware Trojans
- Sub-Transistor ASIC Trojans
- FPGA Trojan
- Key extraction attack
- Auxiliary Stuff
What else can we do with bitstream manipulations?

Hmm, are their simpler ways to extract keys from FPGAs without Trojans?
Set-Up

Can bitstream manipulation of unknown design lead to key leakage?

Can bitstream manipulation of unknown design lead to key leakage?

non-classical set-up: alteration of algorithm (via bitstream)

classical known-plaintext set-up

\[ CT = \text{AES} (k, PT) \]
Bitstream Fault Injections (BiFI)

\[ PT = \text{AES}(k, PT) \]

(surprising) attack strategy
1. manipulate 1st LUT table (e.g., all-zero)
2. configure FPGA
3. send \( PT \)
4. check: Does \( CT \) contain \( k \)?
   if not: GOTO 1 and manipulate next LUT

10-30k LUTs per FPGA
How exactly does the key leak ???

\[ PT \quad CT = AES(k, PT) \]

**Different leakage types (key hypotheses)**
- \( CT = \) roundkey
- \( CT = \) inverted roundkey
- \( CT = PT \) xor roundkey
- ...

**Many LUT manipulations possible**
- all-zero
- all-one
- invert
- upper half of LUT all-zero
- ...

LUT manipulations:
- Configure
- ...
Results for Bitstream Fault Injections (BiFI)

Real world attack
• 16 unknown AES designs (Internet)
• 16 different manipulation rules
• \(\approx 20k\) LUTs
• 3.3 sec for configuring and checking one manipulation

Results
• successful key extraction for every design!
• on average \(\approx 2000\) configurations (\(\approx 2h\))
• works even for encrypted bitstream (w/o MAC)
Conclusion

- Bitstream Fault Injections (BiFI) is a new family of fault attacks
- Malleability of bitstream is major weakness for FPGAs!
- Are there more bitstream-based attacks?
- Details at:
Agenda

- Introduction to Hardware Trojans
- Sub-Transistor ASIC Trojans
- FPGA Trojan
- Key extraction attack
- Auxiliary Stuff
Relevant Conferences

CHES – Cryptographic Hardware & Embedded Systems
Amsterdam, September 9-12, 2018

escar – Embedded Security in Cars
Brussels, November 13-14, 2018
Thank you very much for your attention!

Christof Paar