MAX PLANCK INSTITUTE FOR SECURITY AND PRIVACY

HOW I LEARNED TO STOP WORRYING AND LOVE HARDWARE TROJANS

SUMMER SCHOOL ON REAL-WORLD CRYPTO AND PRIVACY

JUNE 2022

CHRISTOF PAAR

ACKNOWLEDGEMENT

Nils Albartus 

Georg Becker 

Steffen Becker 

Marc Fyrbiak 

Max Hoffmann 

Pawel Swierczynski 

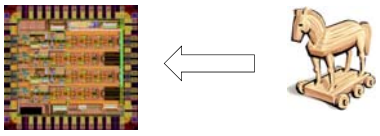
Carina Wiesen 

Agenda

- **Intro & History of Hardware Trojans**
- A Sub-Transistor ASIC Trojan
- FPGA Trojan
- Trojans & Camouflaged Gates
- Hardware Reverse Engineering
- Human Factors in Trojan Design & Detection

HARDWARE TROJANS

"Malicious change to an IC that adds or remove functionality"



Many rather unpleasant "applications"

TROJAN INJECTION & ADVERSARIES SCENARIOS

ing

Hostile hardware blocks
("IP-cores")

during shipment

Built-in by
manufacturer

HISTORICAL PERSPECTIVE: COLD WAR

US WWII
M-209 encryption machine

AB Cryptoteknik
by Boris Hagelin

Cold War
C-52 encryption machine

Crypto AG
by Boris Hagelin

HISTORICAL PERSPECTIVE: COLD WAR

alleged cooperation between *Crypto AG* and intelligence services

Strong indication that C-52 was artificially weakend

HISTORICAL PERSPECTIVE: COLD WAR

1986 Berlin bombing
„La Belle discothèque“

retaliatory air strikes against Libya

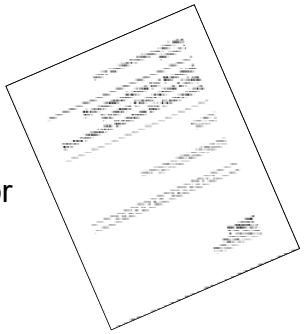
HISTORICAL PERSPECTIVE: RECENT YEARS

- 2017: Law enforcement „going dark“ through strong cryptography

James Comey, FBI

- … crypto backdoors can be „solution“

- Subversion of crypto standards
- Dual ECC random number generator



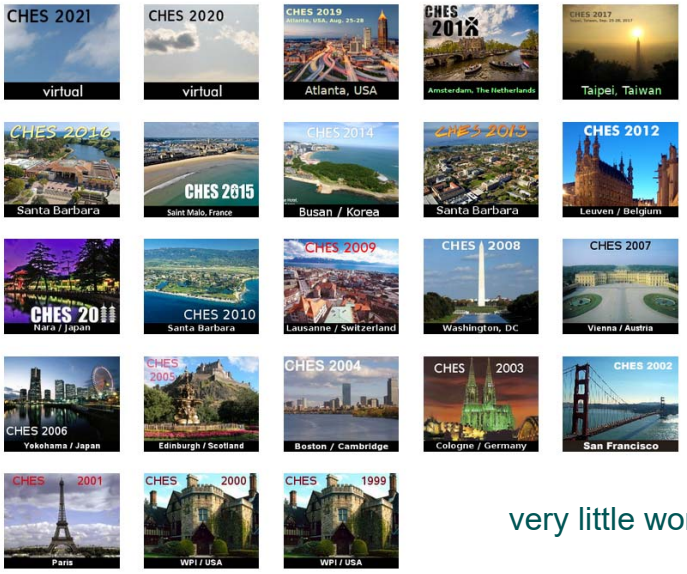
HISTORICAL PERSPECTIVE: 2019

- How trustworthy is foreign-made equipment?

Backdoors in routers ?

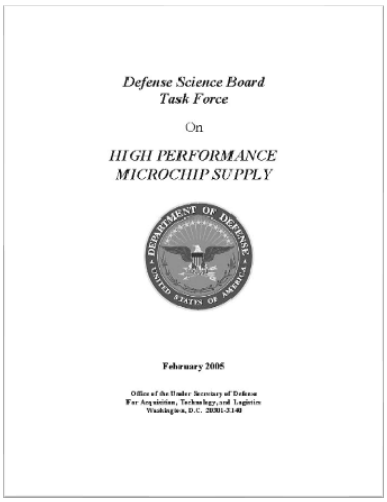
… or in mobile networks?

WHAT ABOUT THE SCIENTIFIC COMMUNITY?



very little work prior to 2005 ….

U.S. DEPARTMENT OF DEFENSE REPORT (2005)



Defense Science Board
Task Force

On

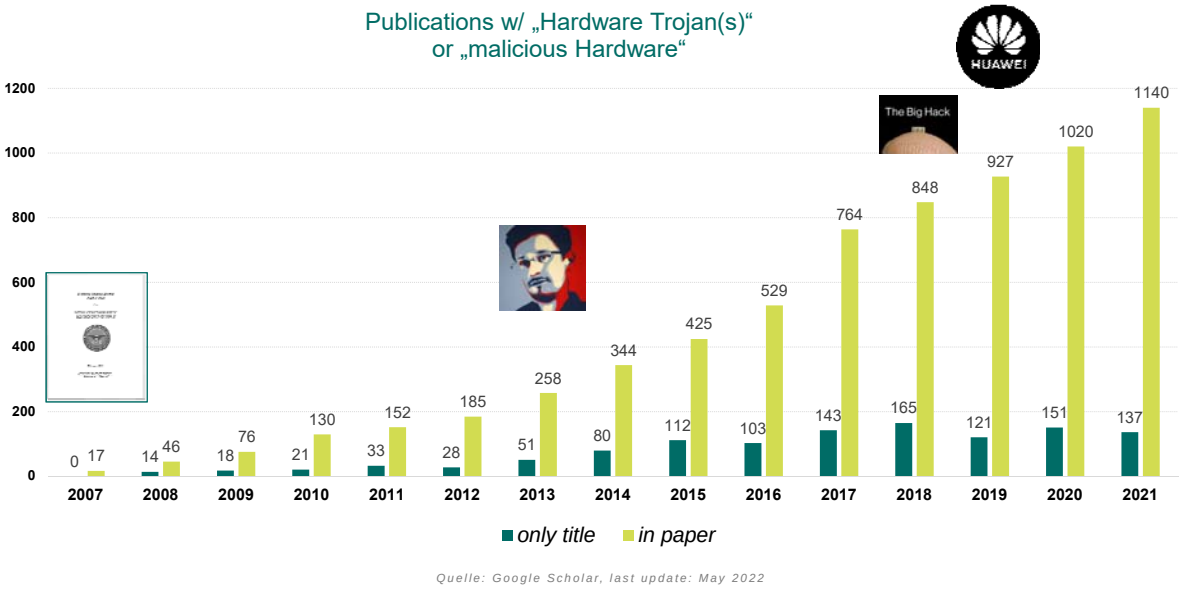
HIGH PERFORMANCE
MICROCHIP SUPPLY

February 2005

Office of the Under Secretary of Defense
For Acquisition, Technology, and Logistics
Washington, D.C. 20301-3140

2005 DoD report triggers research on hardware Trojans

HARDWARE TROJANS AND THE SCIENTIFIC COMMUNITY



Quelle: Google Scholar, last update: May 2022

Agenda

- Intro & History of Hardware Trojans
- **A Sub-Transistor ASIC Trojan**
- FPGA Trojan
- Trojans & Camouflaged Gates
- Hardware Reverse Engineering
- Human Factors in Trojan Design & Detection

WHERE ARE WE WITH “REAL” HW TROJANS?

No true Hardware Trojans in the wild

All examples from academia

Vast majority of publications focus on detection

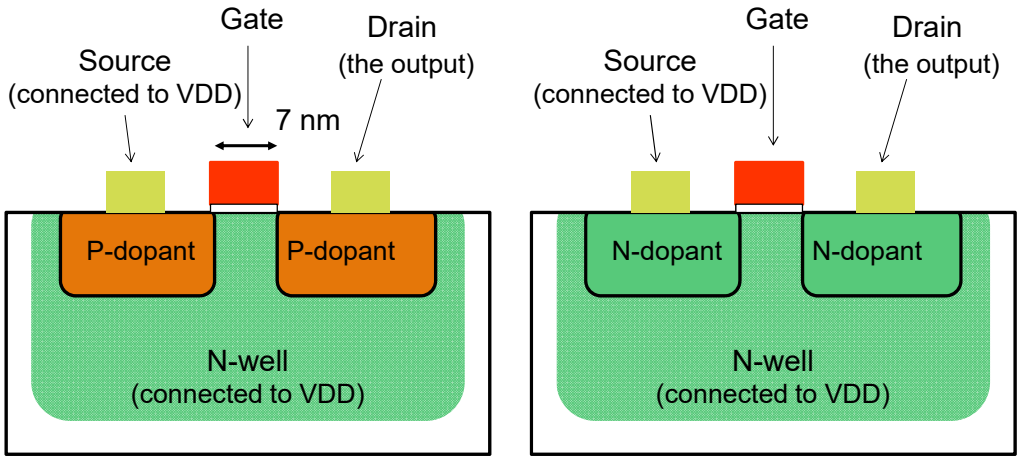
OUR THOUGHTS

1. *Designing* Trojan could be fun too
2. Especially those that go *undetected*

SIMPLE EXAMPLE: INVERTER TROJAN

millions … billions of transistors

PMOS TRANSISTOR TROJAN

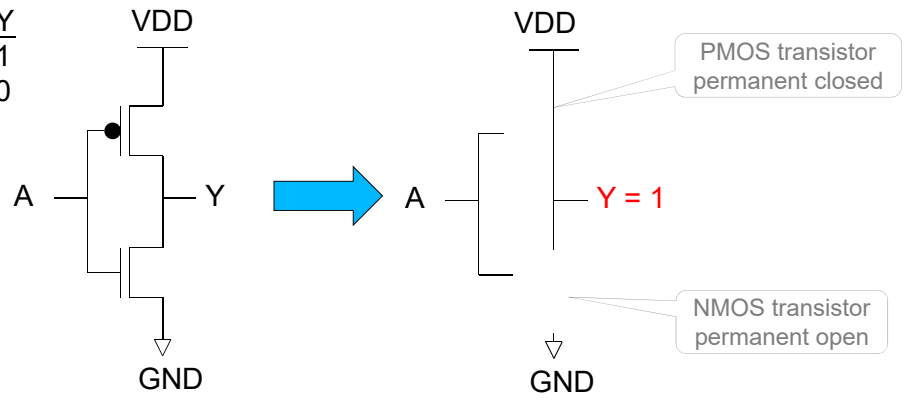


Unmodified PMOS transistor

Trojan transistor “always 1”

"ALWAYS ONE" TROJAN INVERTER

A	Y
0	1
1	0



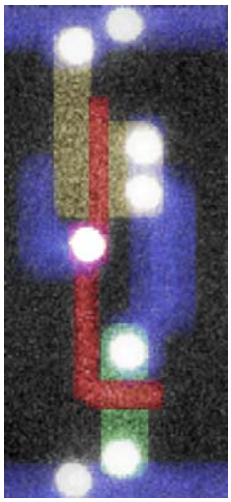
Q1: Can the manipulation be detected?

Q2: How to build a useful Trojan from here?

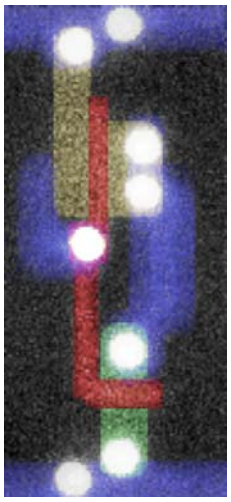
DETECTION: SILICON VIEW ON TROJAN INVERTER

Which one has the Trojan?

Original Inverter



"Always 1" Trojan



Dopant changes (very ?) difficult to detect using optical inspection!

“SMALL” REMAINING QUESTION

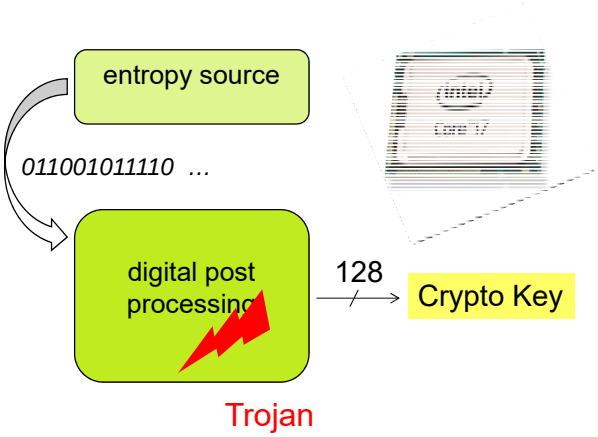
Q2: Can we build a **meaningful** Trojan using dopant modifications that passes functional testing?

A REAL-WORLD TRUE RANDOM NUMBER GENERATOR (TRNG)

… random numbers generate cryptographic keys for

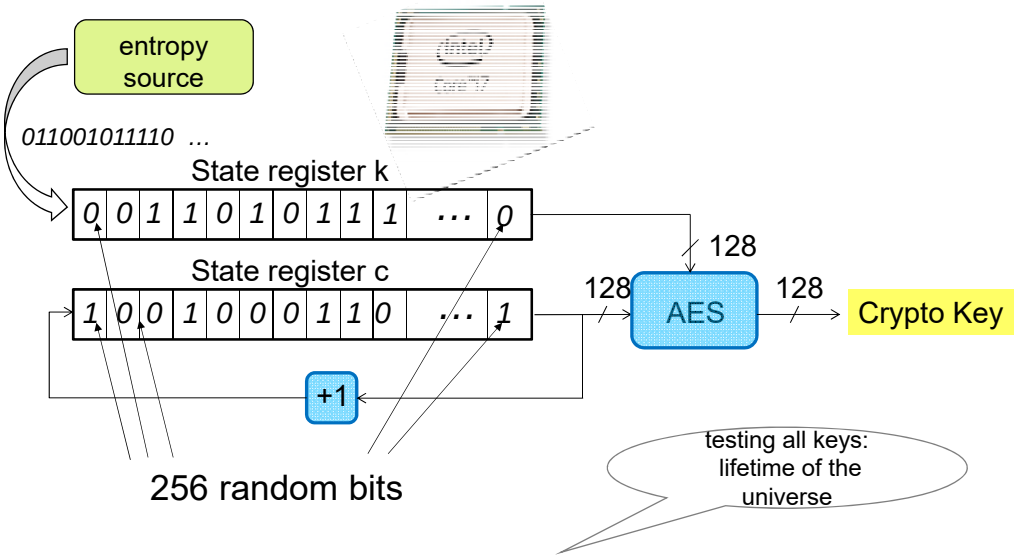
- secure web browsing
- email encryption
- document certification
- …

1-SLIDE VERSION OF TRNG TROJAN



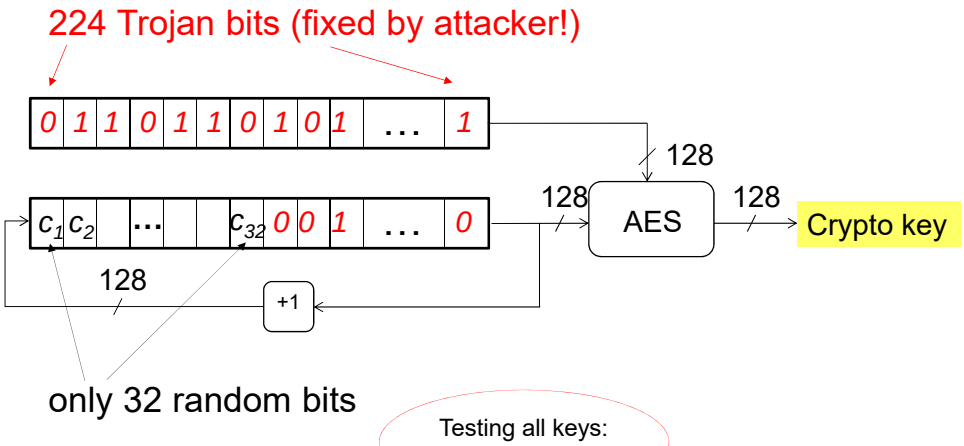
- regular: 2^{128} keys
- with Trojan: 2^{32} keys

INSIDE THE RANDOM NUMBER GENERATOR



- 1,000,000,000,000,000,000,000,000,000,000,000,000,000,000,000 possible crypto keys

TROJAN RANDOM NUMBER GENERATOR



Testing all keys: few seconds

- 1,000,000,000,000,000,000,000,000,000,000,000,000,000 possible crypto keys

CONCLUSION

Slashdot · SPIEGEL ONLINE ·  · ars technica · COMPUTERWORLD

- Meaningful hardware Trojans are possible without extra logic
- Many detection techniques don't guarantee a Trojan free design!
- 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

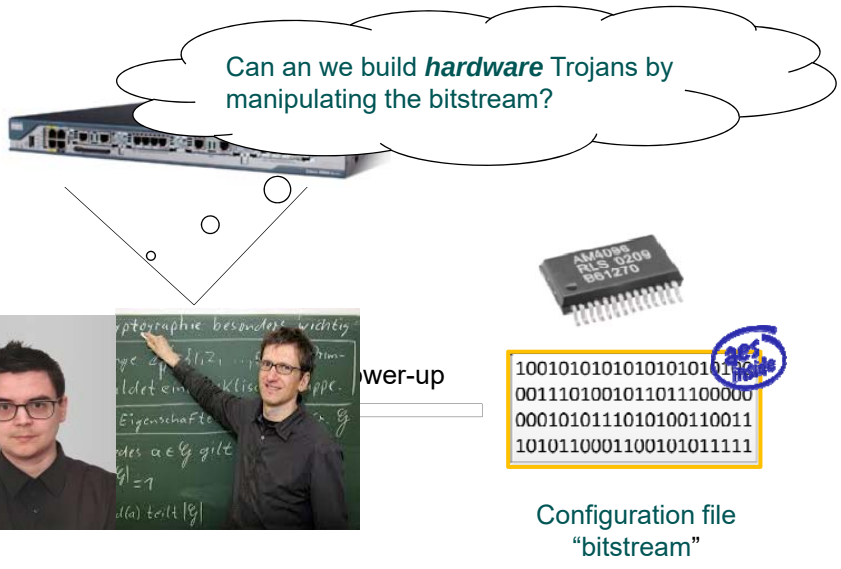
- Intro & History of Hardware Trojans
- A Sub-Transistor ASIC Trojan
- **FPGA Trojan**
- Trojans & Camouflaged Gates
- Hardware Reverse Engineering
- Human Factors in Trojan Design & Detection

FPGAS = RECONFIGURABLE HARDWARE … ARE WIDELY USED

world market: >5b devices

2 dominant companies

CONFIGURATION DURING POWER-UP



ALGORITHM SUBSTITION ATTACK ON AES

Idea: Replace regular (strong) crypto alg. with a weak one

trojanizing AES:

1. Find S-Boxes
2. Replace with weak S-Boxes

100101010101010101010
001110100101101110000
000101011101010011001
101011000110010101111

AES

most widely used cipher worldwide

100101010101010101010
001110100101101110000
000101011101010011011
101011000110010101111

AES^{T}

trojanized version of AES, allowing key extraction etc.

AES DETECTION

luckily, S-box values are **very** unique

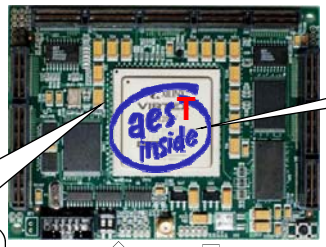
8 different real-world AES implementations

Impl.	Architecture	AES	LUTs with S-box logic	S-boxes in memory	Detection
#1	Round-based	128	(16+4)·32 = 640	no	100 %
#2	$\frac{1}{4}$ Round	128	0	yes	100 %
#3	$\frac{1}{4}$ Round	192	0	yes	100 %
#4	$\frac{1}{4}$ Round	256	0	yes	100 %
#5	Round-based	128	(0+4) · 32 = 128	yes	100 %
#6	Round-based	128	0	yes	100 %
#7	Round-based	128	0	yes	100 %
#8	Round-based	128	(16+4)·32 = 640	no	100 %

TABLE IV: Overview of evaluated AES implementations

ALGORITHM SUBSTITUTION ATTACK AND ITS IMPLICATIONS

Attack scenario:
Adversary has (temporarily) physical access to devices



2. **Trojan AES** is configured

1. Inject **weak S-boxes** in bitstream

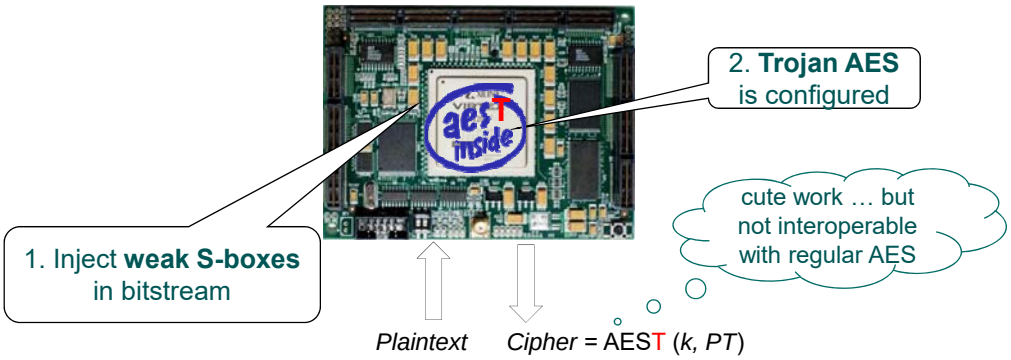
cute work … but not interoperable with regular AES

Plaintext *Cipher* = AEST (*k*, *PT*)

Storage encryption (cloud etc.)

- Encryption & decryption use same implementation!
- Attacker can recover plaintext without access to *k*
- *Non-suspicious: data still looks encrypted*

ALGORITHM SUBSTITUTION ATTACK AND ITS IMPLICATIONS



Key-extraction attack

- Assumes temporary device access
- switch S-box and recover key *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 & USENIX 2020)
- details at:
  Swierczynski, Fyrbiak, Koppe, P, *FPGA Trojans through Detecting and Weakening of Cryptographic Primitives*. IEEE TCAD 2015.

Agenda

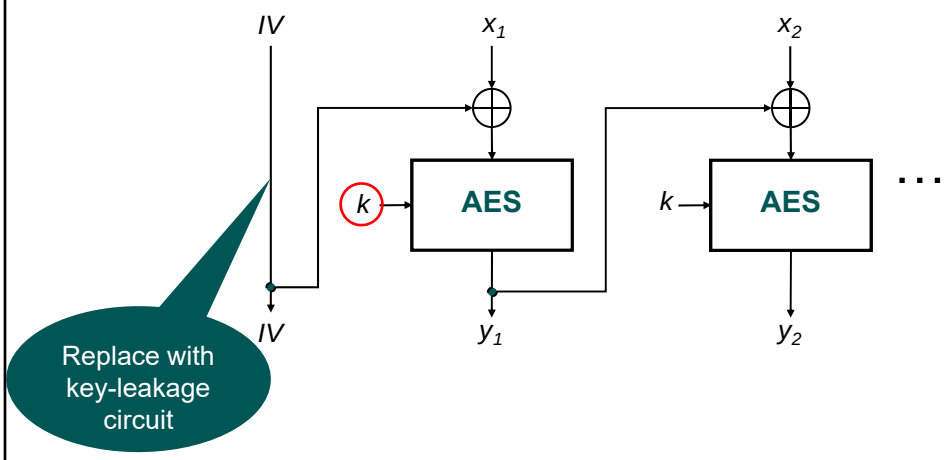
- Intro & History of Hardware Trojans
- A Sub-Transistor ASIC Trojan
- FPGA Trojan
- **Trojans & Camouflaged Gates**
- Hardware Reverse Engineering
- Human Factors in Trojan Design & Detection

OUR THOUGHTS

How can we leak an encryption key without raising suspicion?

… let's try to hide the key in the IV?

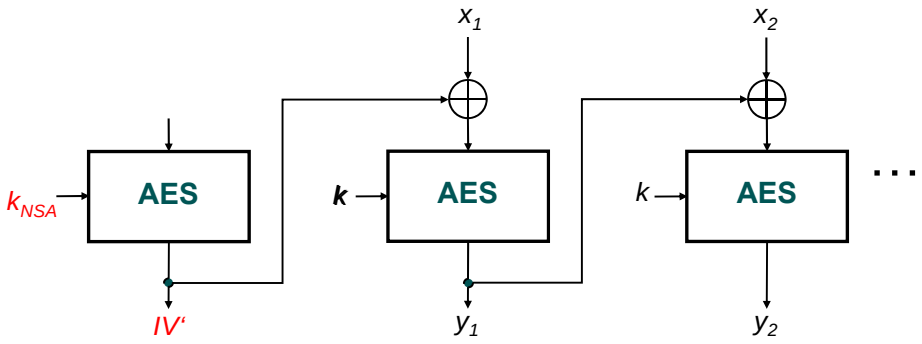
ALGORITHMIC SUBSTITUTION ATTACK ON CBC MODE



Goal 1: Leak encryption key k

Goal 2: Control exploitability

ALGORITHMIC SUBSTITUTION ATTACK ON CBC MODE



Attack

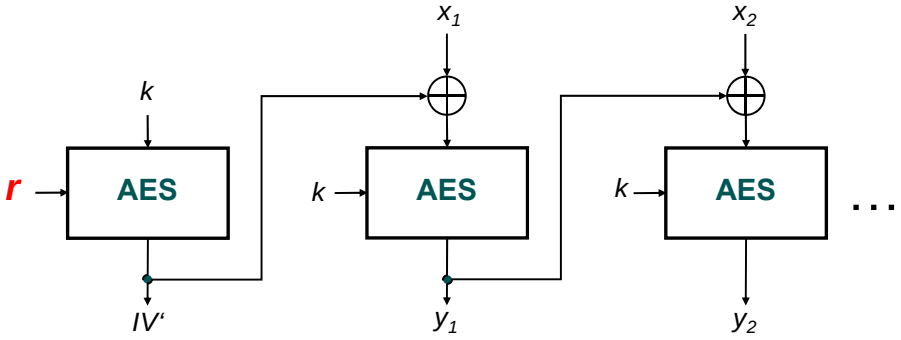
Key recovery: $k = AES_{K_{NSA}}^{-1}(\mathrm{IV}')$

Decrypt payload: $x_i = AES_K^{-1}(y_i)$

Real-world problem: *IV‘* doesn‘t change (for fixed *k*)

... let‘s introduce randomness

VARIANT WITH RANDOM (BUT LOW-ENTROPY) KEY



Attack

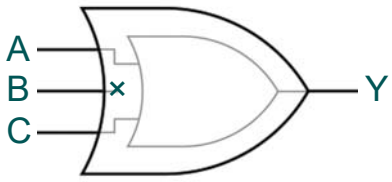
$$r \in_R \{0, \dots, 2^\lambda - 1\}$$

FOR $i = 0\ TO\ 2^\lambda - 1$:

1. $k = AES_r^{-1}(\mathrm{IV}')$
2. $x_i = AES\_k(y_i)$

(Big) Question: How can we hide the attack circuit?

Basic Elements: Camouflaged Gates with Dummy Inputs

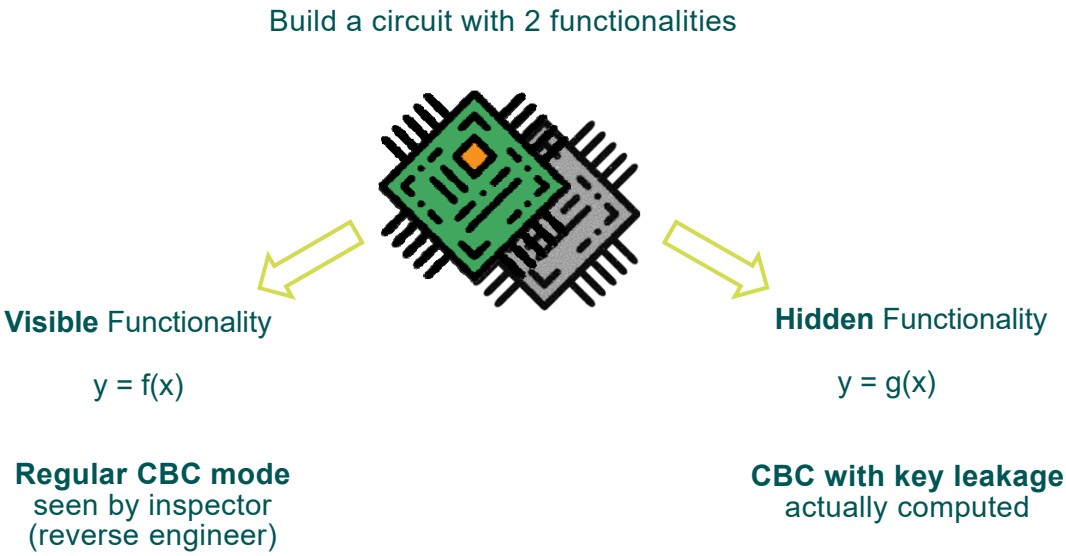


- What a reverse engineer sees:
  Y = A or B or C
- What is actually computed:
  Y = A or C

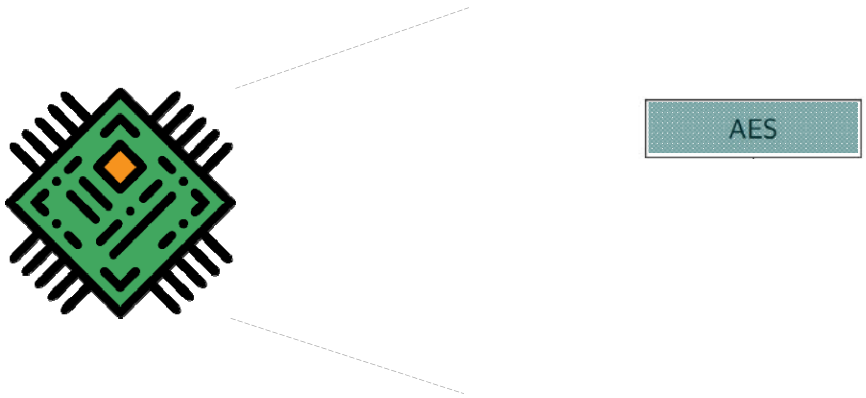
Scientific and commercial instantiations are available, e.g., [1,2,3].

[1] Georg T. Becker, et al. "Stealthy dopant-level hardware trojans." *CHES 2013*.
[2] Bicky Shakya, et al. "Covert gates: Protecting integrated circuits with undetectable camouflaging." *CHES 2019*.
[3] Rambus. SypherMedia Library (SML) Circuit Camouflage Technology.

Doppelganger in a Nutshell

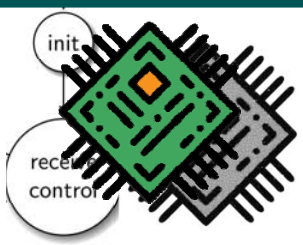


Cryptographic Coprocessor: Design



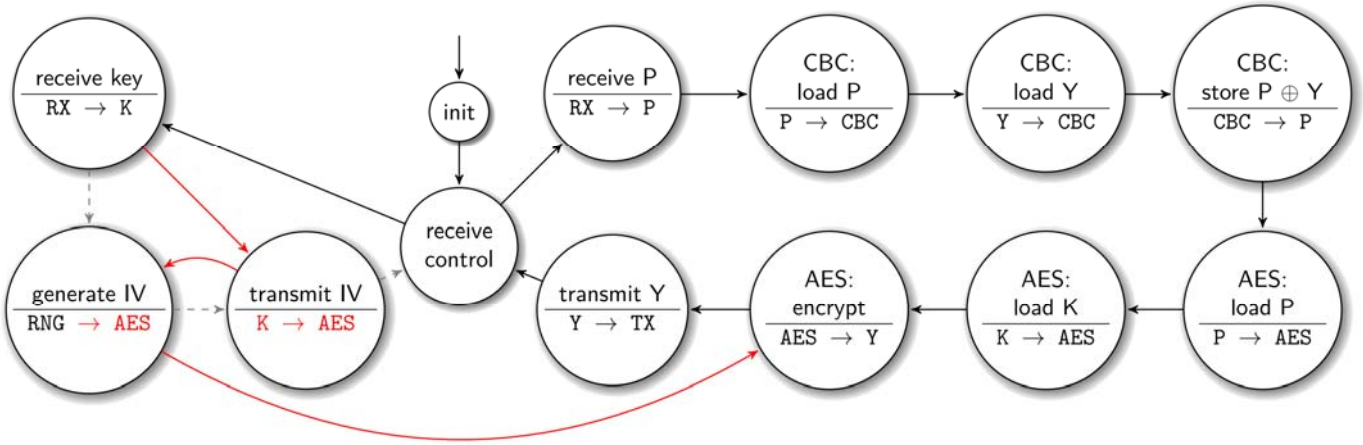
CBC Coprocessor: Finite State Machine

let‘s doppelganger the FSM



Trojanized FSM: Hidden Functionality

only 6 signals have to be connected to dummy inputs.



virtually no overhead

- original: 16,776 … 16,686 gate eq
- Doppelganger: 16,683 gate eq

Conclusion

- **scary** application of camouflaged gates
- but: Doppelganger has also constructive use in **hardware obfuscation**
- all existing static analysis method for Trojan detection **fail**
- non-invasive **detection** *might* be possible:
  1. fine-grain time measurement of IV generation
  2. if this very Doppelganger is expected, the design can be tested for IV collisions
     - e.g. $\lambda = 32 \quad \rightarrow 2^{16}$ tests until collission
- details:
  Hoffmann, P: *Doppelganger Obfuscation — Exploring the Defensive and Offensive Aspects of Hardware Camouflaging*. CHES 2021

Agenda

- Intro & History of Hardware Trojans
- A Sub-Transistor ASIC Trojan
- FPGA Trojan
- Trojans & Camouflaged Gates
- **Hardware Reverse Engineering**
- Human Factors in Trojan Design & Detection

Hardware Reverse Engineering (HRE)

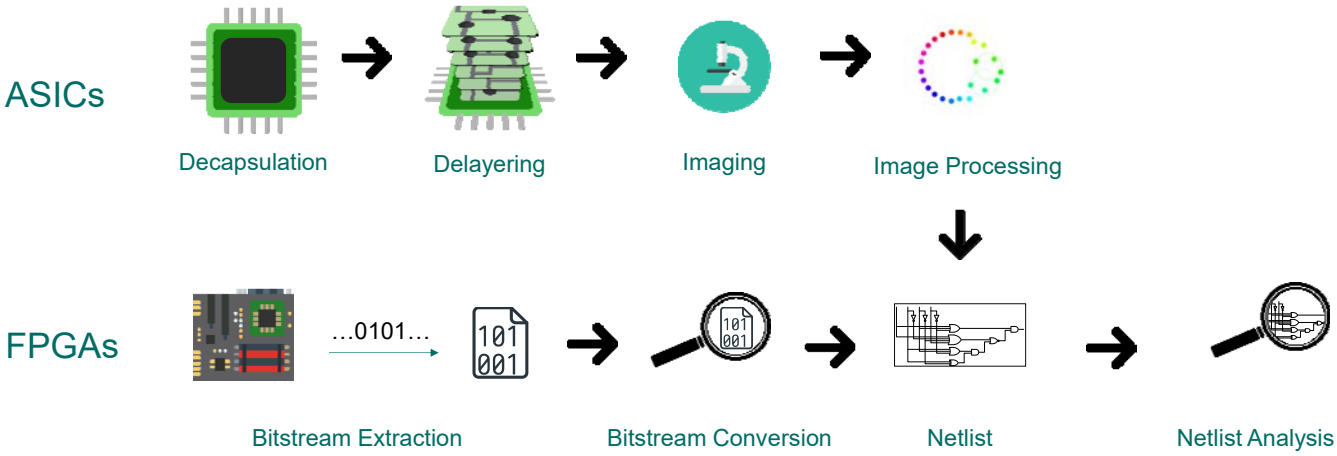
„Reverse Engineering is the process by which a man-made object is deconstructed to reveal its design and architecture or to extract knowledge from the object. “

very relevant in Trojan context:

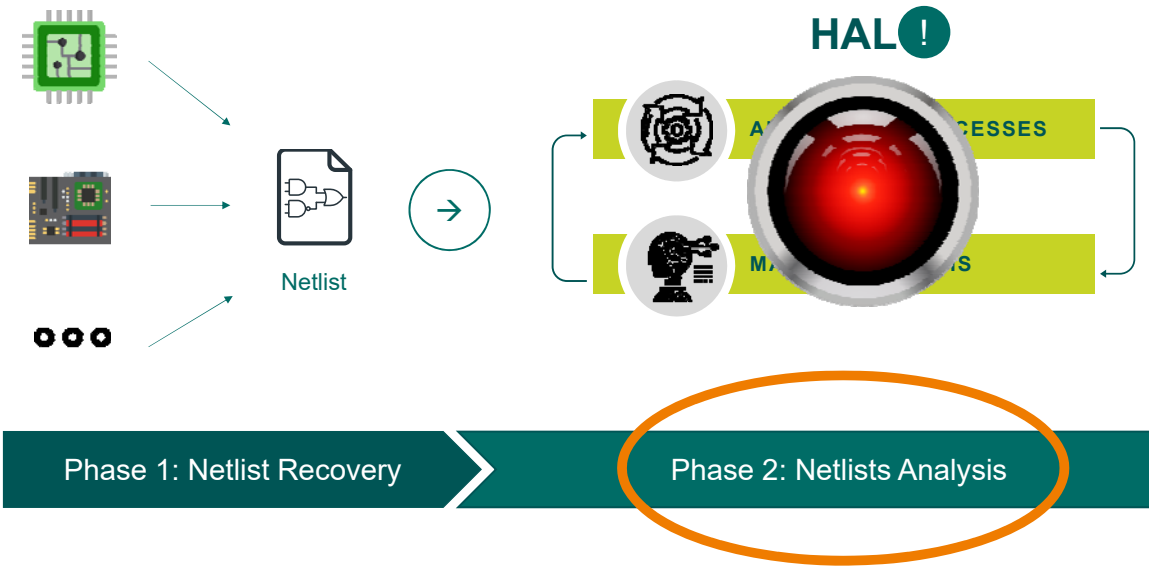
- Constructively: verification of **Trojan freeness**
- Destructively: **Trojan injection** in 3rd party design

„Malicious Foundry“
cf. DoD Report 2005

STEPS IN HARDWARE REVERSE ENGINEERING



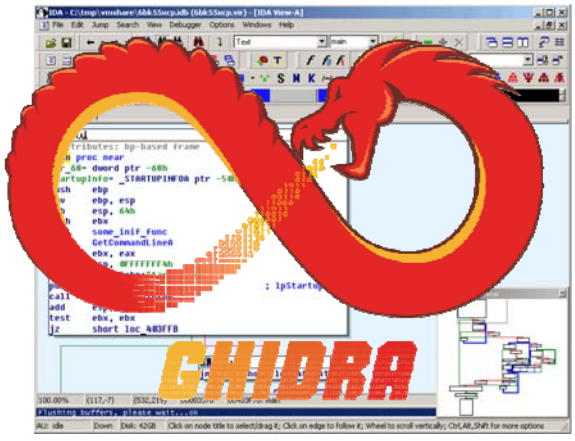
HIGH-LEVEL VIEW ON HARDWARE-RE



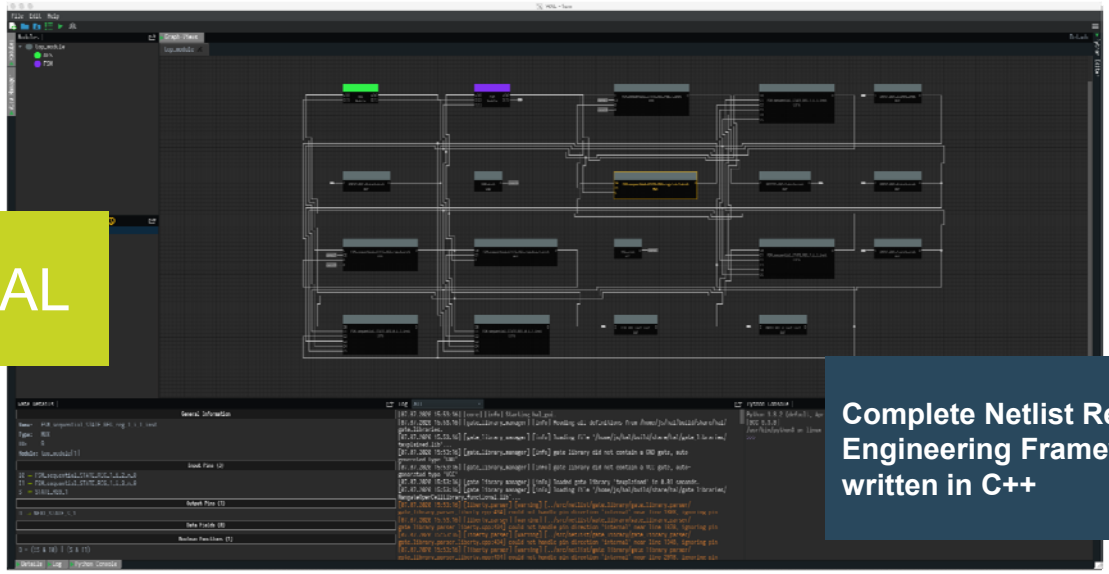
SOFTWARE REVERSE ENGINEERING

- **Software reverse engineering is very popular and has a very active research community**
- **Most popular tool: IDA Pro**
  - Complete framework for binary analysis
  - Modular (plugins)
- **New(ish): GHIDRA by the NSA**
- **Open-source software available**

- **Situation for HW RE quite different…**

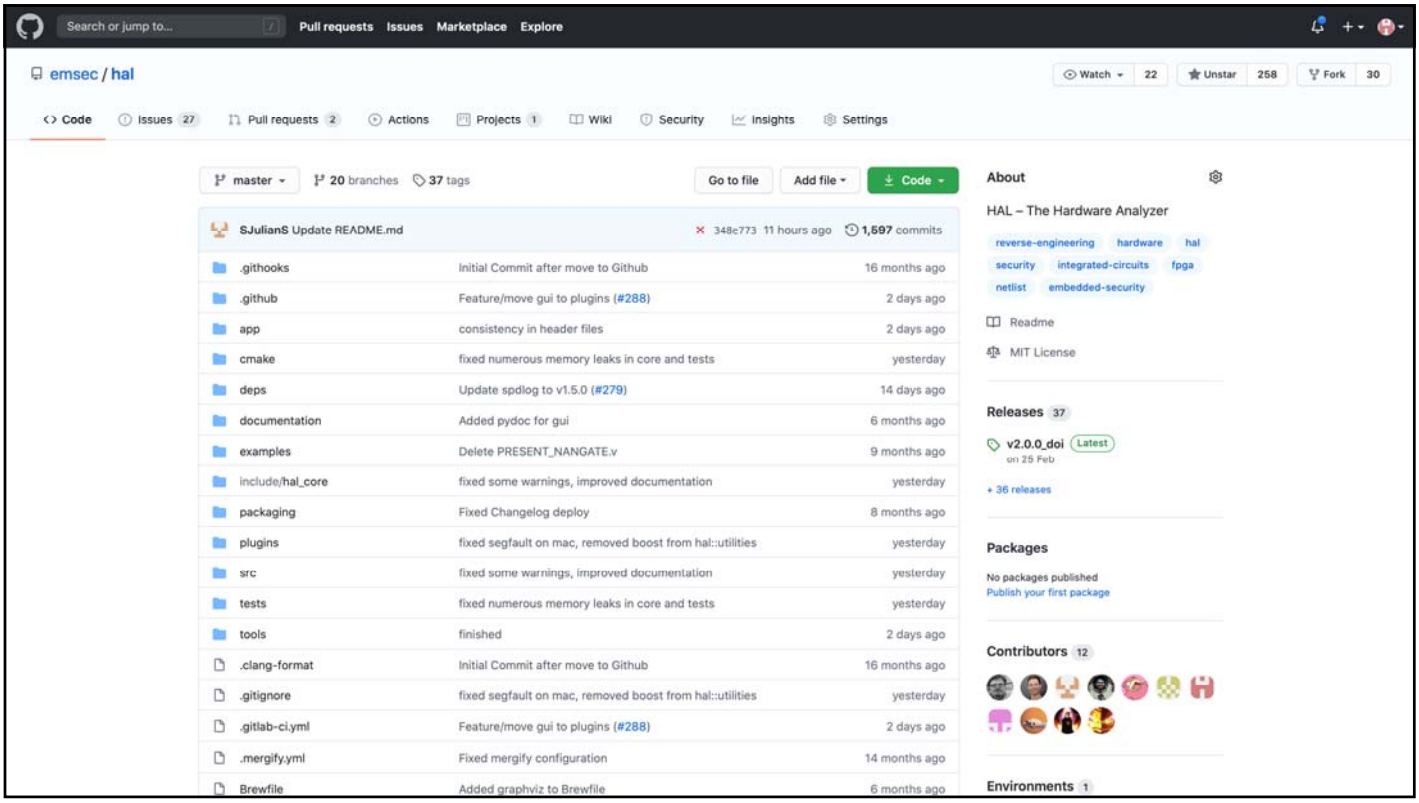


HAL



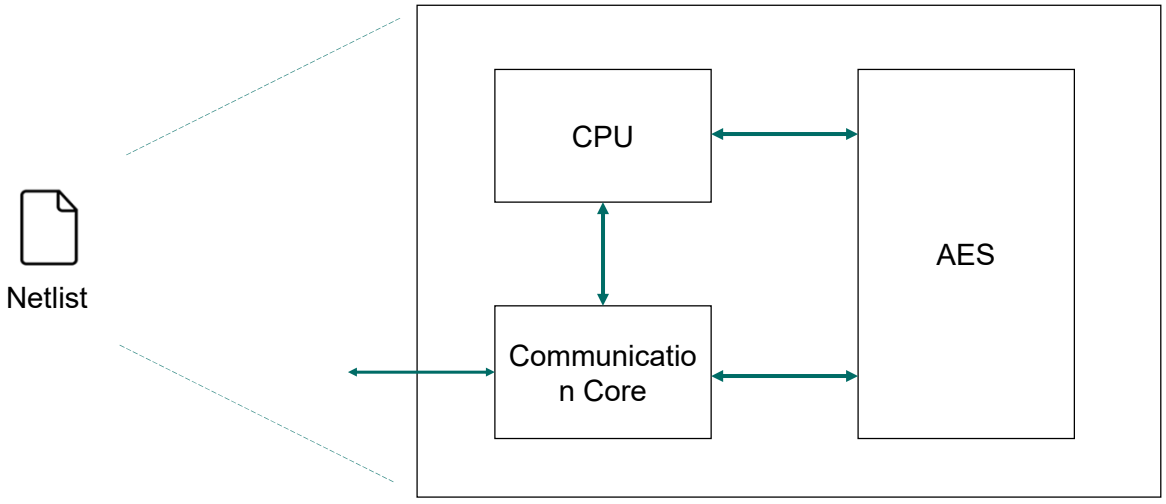
Complete Netlist Reverse Engineering Framework written in C++

HAL & DANA | NILS ALBARTUS, MAX HOFFMANN | 26.08.20 30



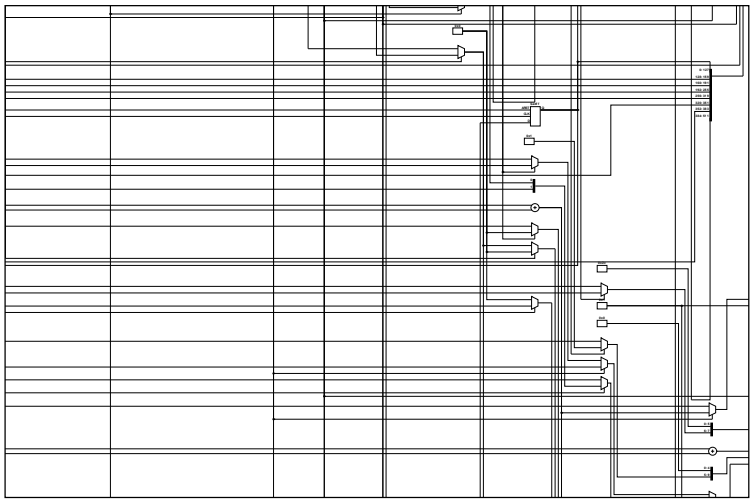
WHY WE *REALLY* NEED HAL: NETLIST ANALYSIS

How you might think it looks like…



HW REVERSE ENGINEER RECOVERS *SEA-OF-GATES*

Netlist



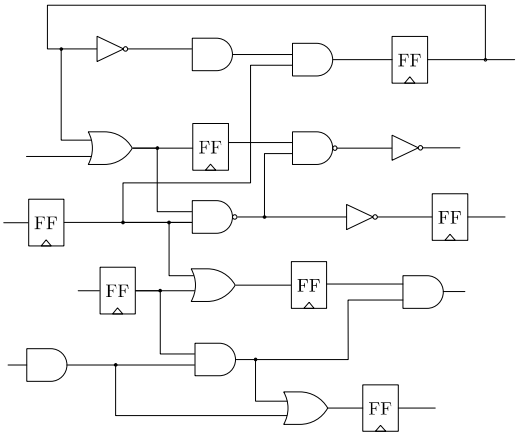
OUR THOUGHTS

*Can HW Reversing be **automated** using HAL?*

DATAFLOW ANALYSIS – STARTING POINT

- Unprocessed netlist = incomprehensible sea of gates
- What does it do?
- Where to start?

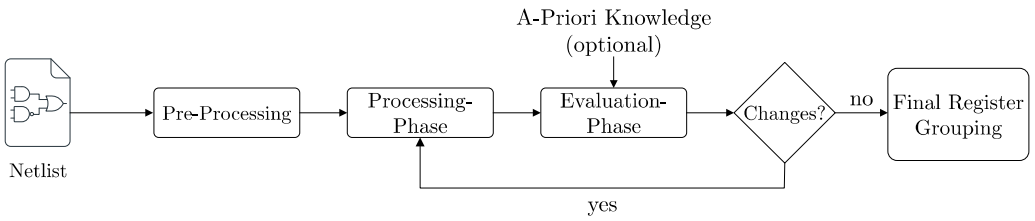
DATAFLOW ANALYSIS – GENERAL IDEA



DANA IN A NUTSHELL

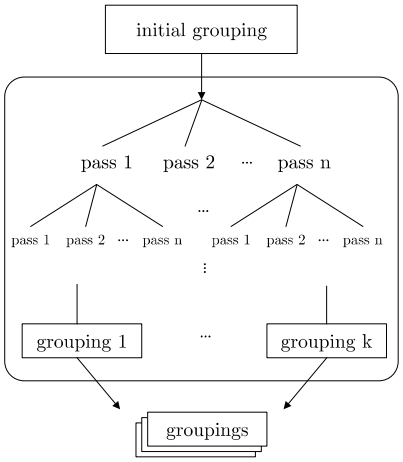
1. Try to identify registers
2. Ignore combinational logic
3. Independently combine multiple small and simple metrics
4. The data decides by itself what the final output should look like

DANA – GENERAL WORKFLOW



A-Priori Knowledge:
Expected Register Size

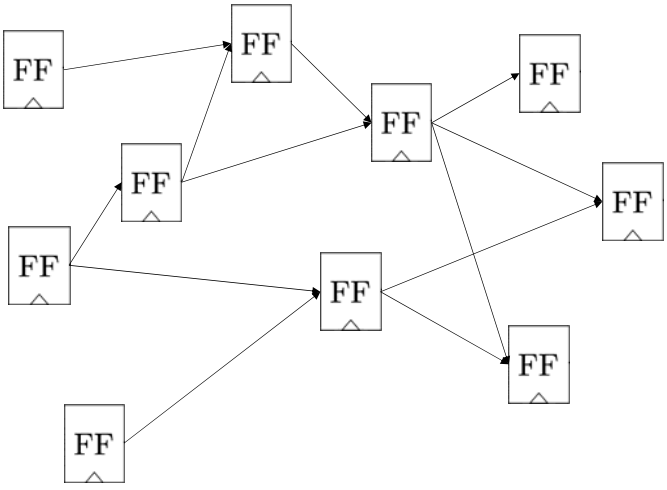
DANA – PROCESSING PHASE

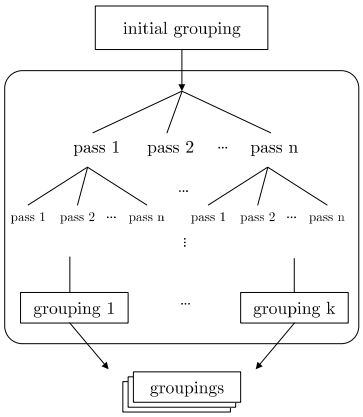


Pass:

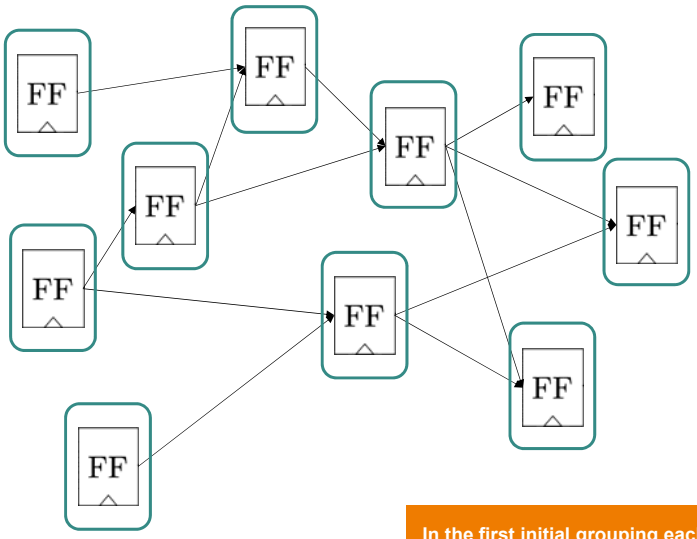
Applies metric to group FFs into register

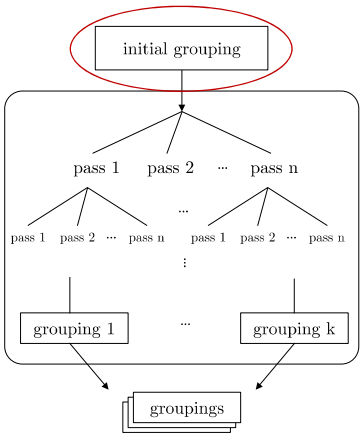
DANA – PROCESSING PHASE





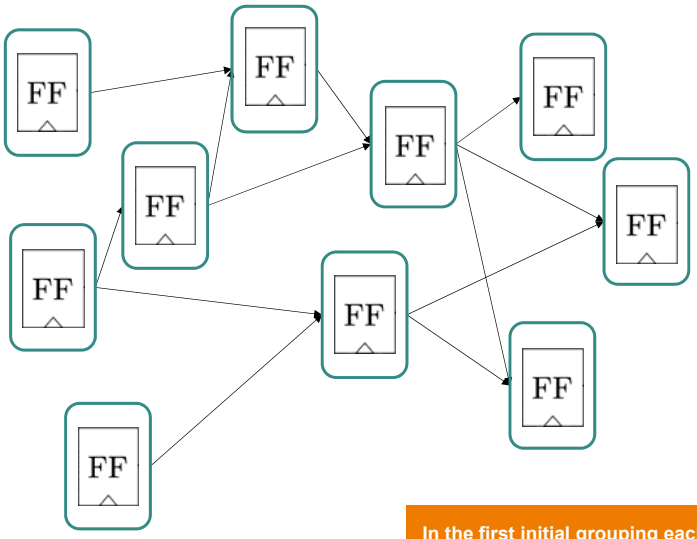
DANA – PROCESSING PHASE

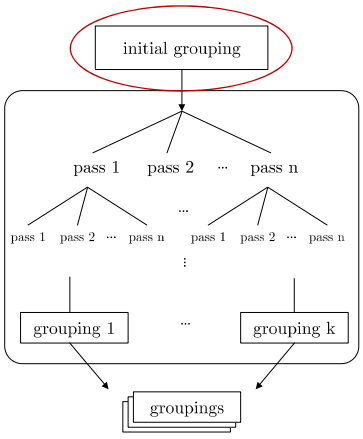




In the first initial grouping each FF is in its own group/register

DANA – PROCESSING PHASE

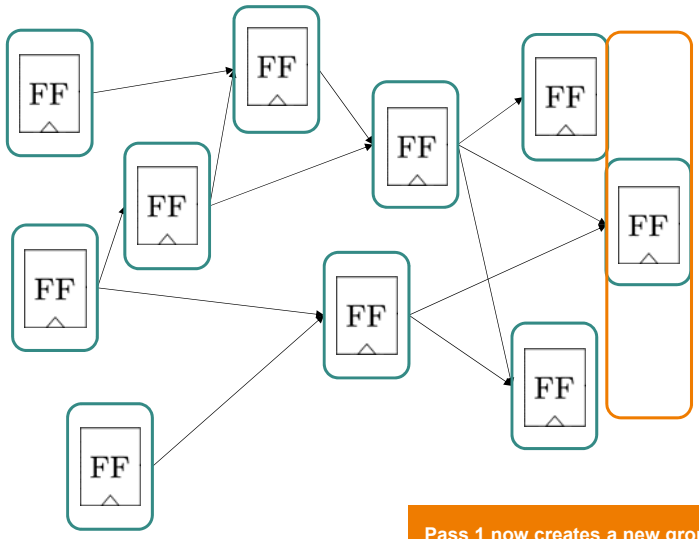


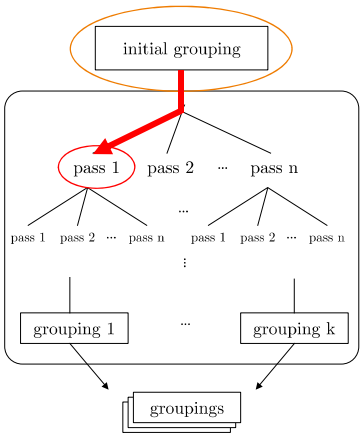


In the first initial grouping each FF is in its own group/register

HAL & DANA | NILS ALBARTUS, MAX HOFFMANN | 26.08.20 30

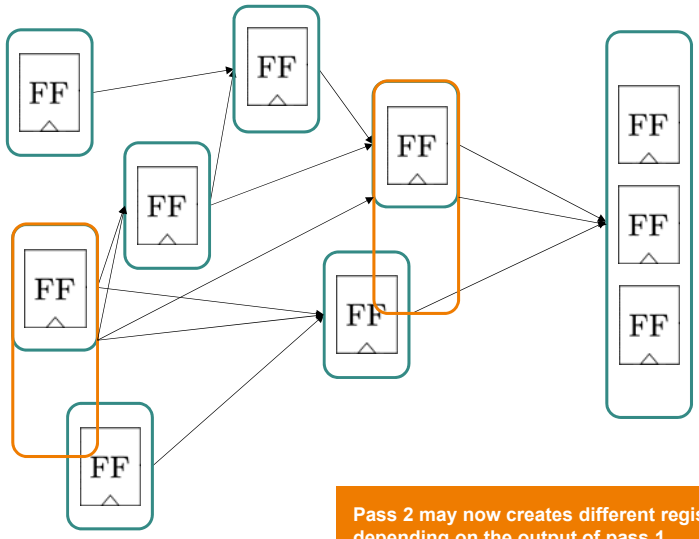
DANA – PROCESSING PHASE

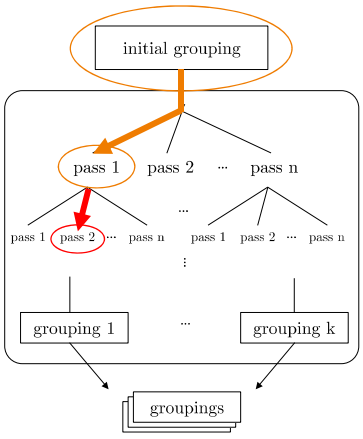




Pass 1 now creates a new grouping, based on the input grouping

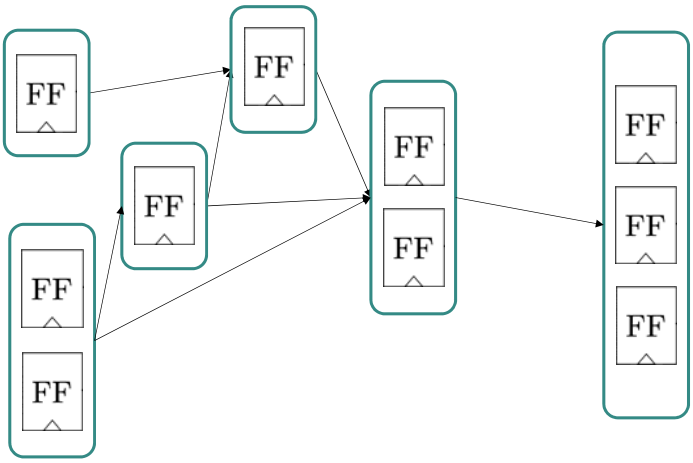
DANA – PROCESSING PHASE

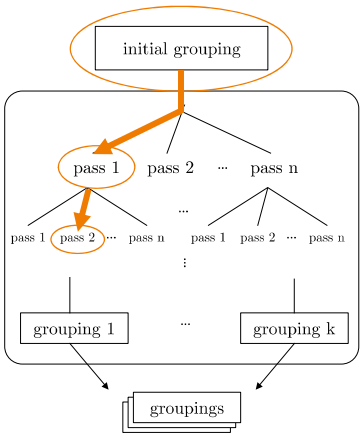




Pass 2 may now creates different register(s) depending on the output of pass 1

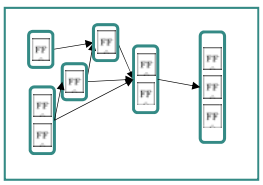
DANA – PROCESSING PHASE



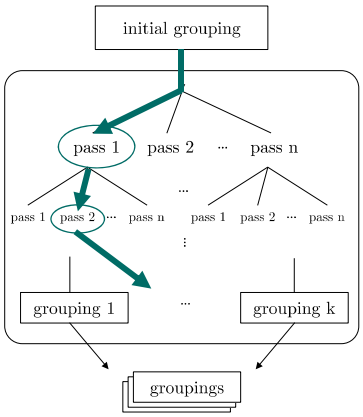


Pass 2 may now creates different register(s) depending on the output of pass 1

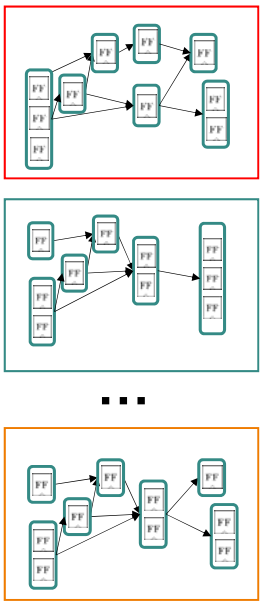
DANA – PROCESSING PHASE



- grouping 2



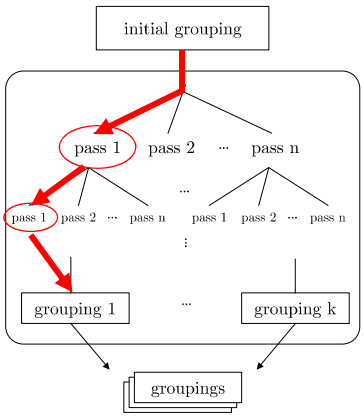
DANA – PROCESSING PHASE



- grouping 1
- grouping 2

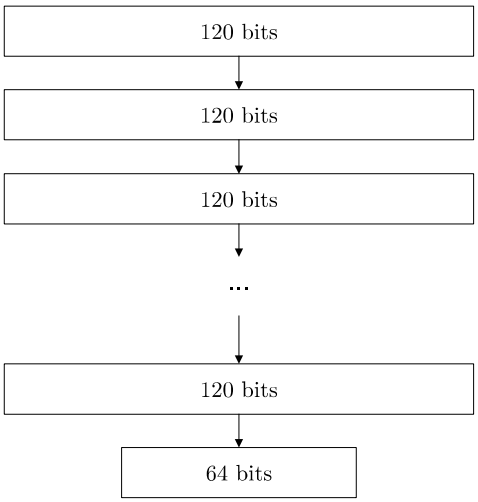
...

- grouping k



APPLICATION OF DANA TO *DES* ENGINE

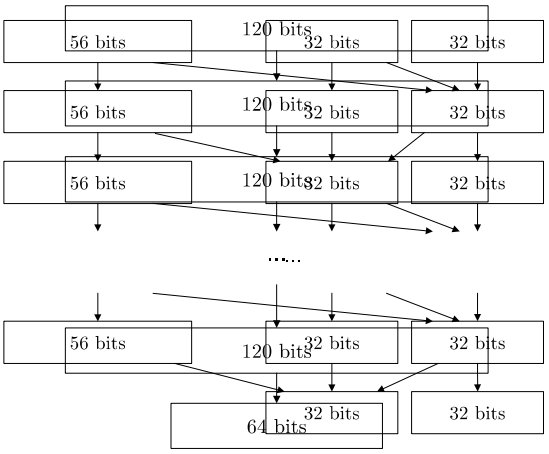
- **DANA generates graphs**
- **Boxes = register**
- **Arrows = logic between two registers**



DANA'S SECRET WEAPON

DES DETECTION IN *STEERED MODE*

- **Reverse Engineer knows/assumes register sizes**
- **These sizes will be prioritized in the voting**
- **For DES:**
  - Key: 56-bit
  - Rounds: 2x 32-bit



CASE STUDY: *OPENTITAN*

"OpenTitan is the first open source project building a transparent, high-quality reference design and integration guidelines for silicon root of trust (RoT) chips."

Google ETH Zürich lowRISC ...

Our Goal: Identify cryptographic primitives in OpenTitan

HYPOTHESIS OPENTITAN

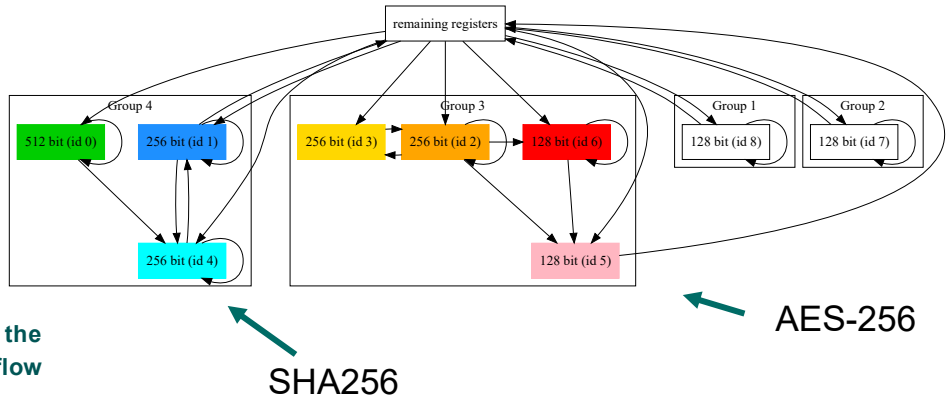
Module	Expected Register Sizes
AES-256	256 (Key), 128 (State)

→ Steering DANA to registers of size: 512, 256, 128

OPENTITAN – RESULTS

DANA identified the following registers:

- **1 x 512-bit**
- **4 x 256-bit**
- **4 x 128-bit**

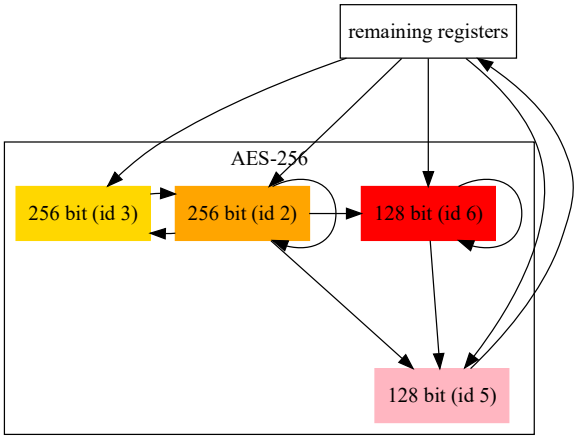


AES-256

SHA256

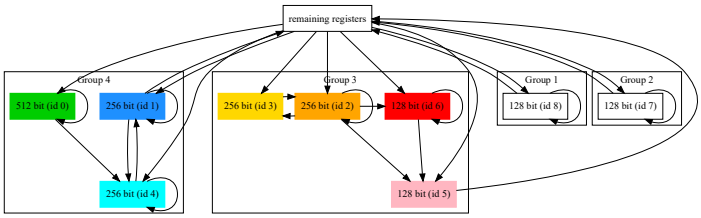
- **There should be a connection between the modules in the dataflow graph**

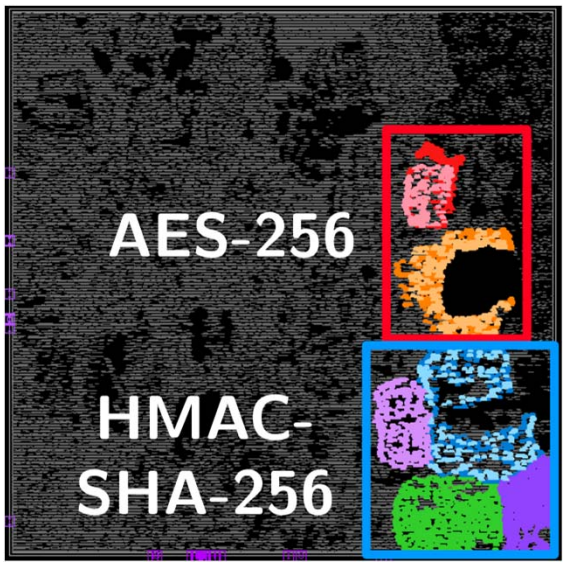
A CLOSER LOOK ON AES-256



- **Red – 128-bit State Register**
  - Only influenced by itself (round functions) and the key
- **Orange – 256-bit Key Register**
  - Updates itself (key-scheduele)
  - Influences the state register
- **Pink – 128-bit Output Register**
  - Influenced by state register
  - Only connection outside of module

APPLYING FINDINGS TO PLACED NETLIST





Conclusion

- DANA allows **module identification** in unknown designs
- **grouping flip-flops into registers** can give much insights into netlist
  - (… while complitely ignoring Boolean logic)
- works well in **data-path driven architectures**
- **not a silver bullet** for all circuits
- more info:

  Albartus, Hoffmann, Temme, Aziel, P: *DANA – Universal Dataflow Analysis for Gate-Level Netlist Reverse Engineering*. CHES 2020

Agenda

- Intro & History of Hardware Trojans
- A Sub-Transistor ASIC Trojan
- FPGA Trojan
- Trojans & Camouflaged Gates
- Hardware Reverse Engineering
- **Human Factors in Trojan Design & Detection**

BUT WAIT, HOW DO WE PROCEED AFTER DANA FOUND MODULES?

"Major role in reverse engineering humans play"

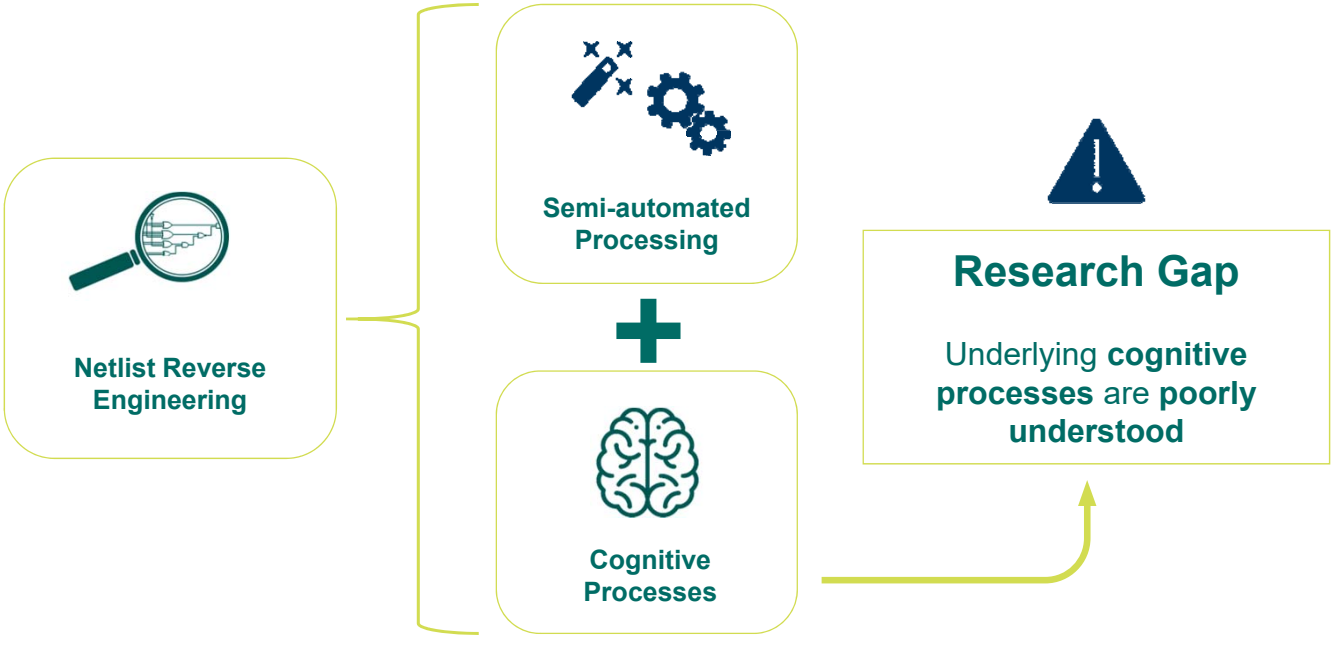
OUR THOUGHTS

Let's try to study human factors in HRE!





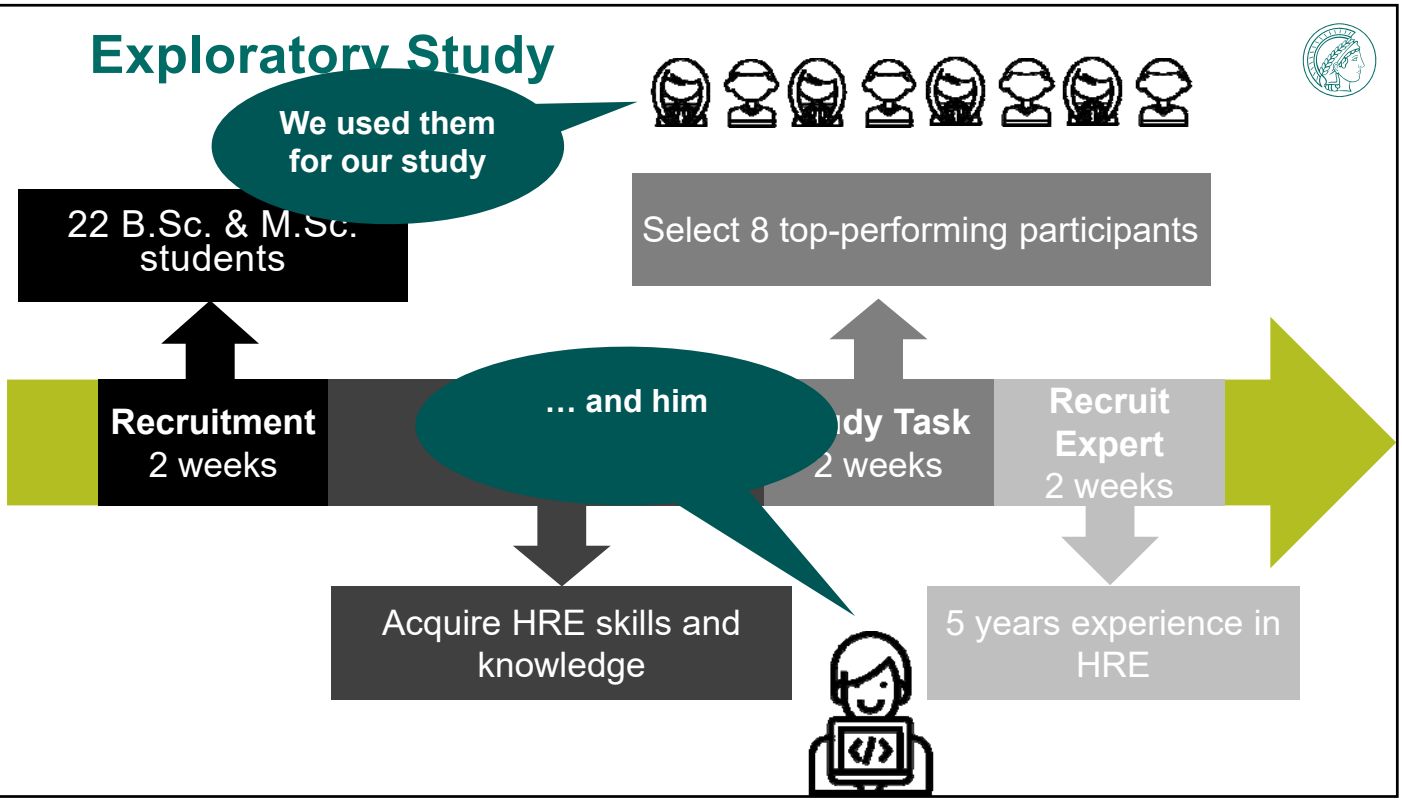
Model of HRE and Research Gap



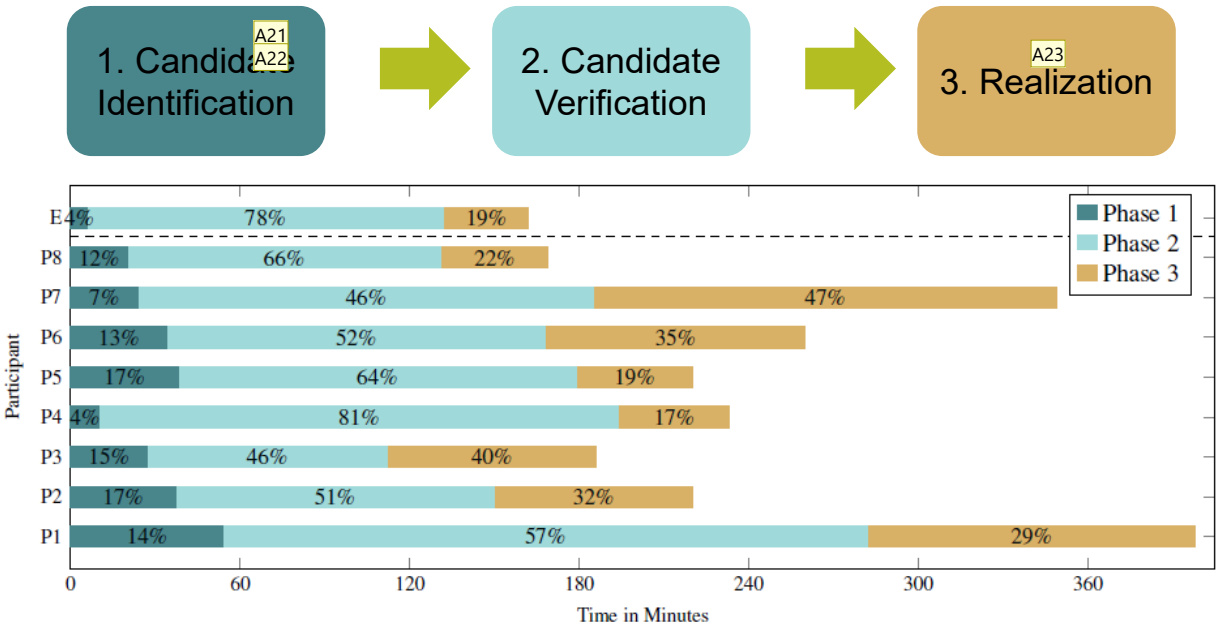
Research Questions

- RQ1: Which are the crucial **phases of human sense-making** during HRE?
- RQ2: Which **strategies** distinguish more and less efficient reverse engineers?
- RQ3: Which **cognitive prerequisites** play a role for the success of HRE?
- RQ4: Which hypotheses can be derived for **cognitive obfuscation**?

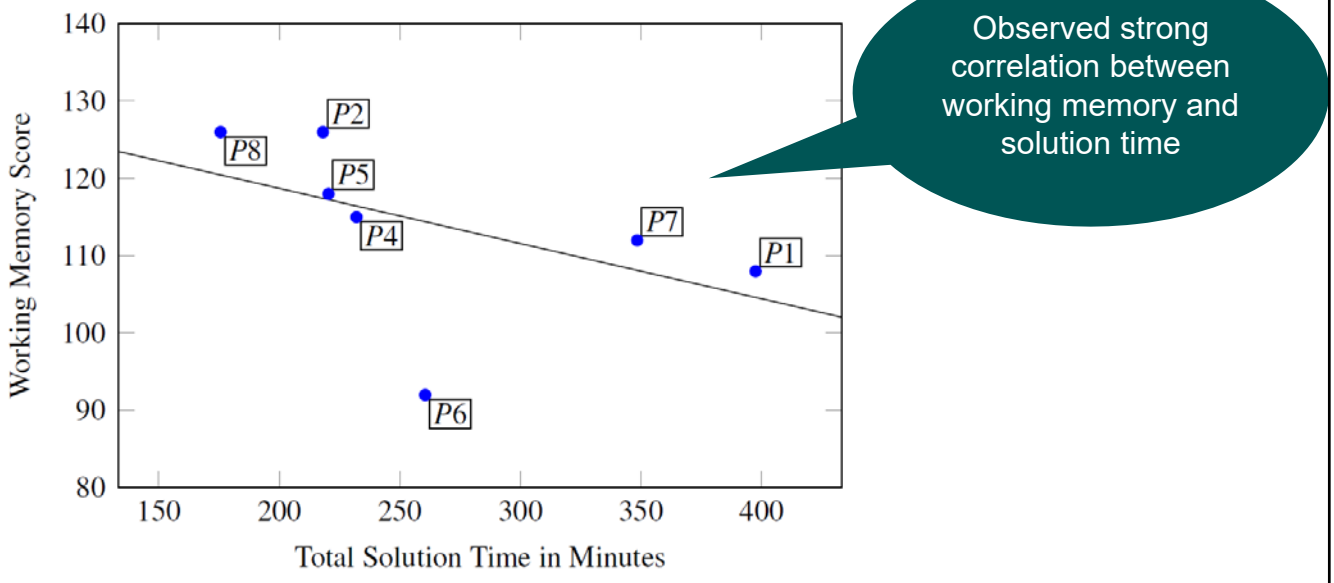
Exploratory Study



RQ1: 3 Human Sensemaking Phases in HRE



RQ3: Cognitive Factor Working Memory



Disclaimer: only descriptive analysis; further research needed!

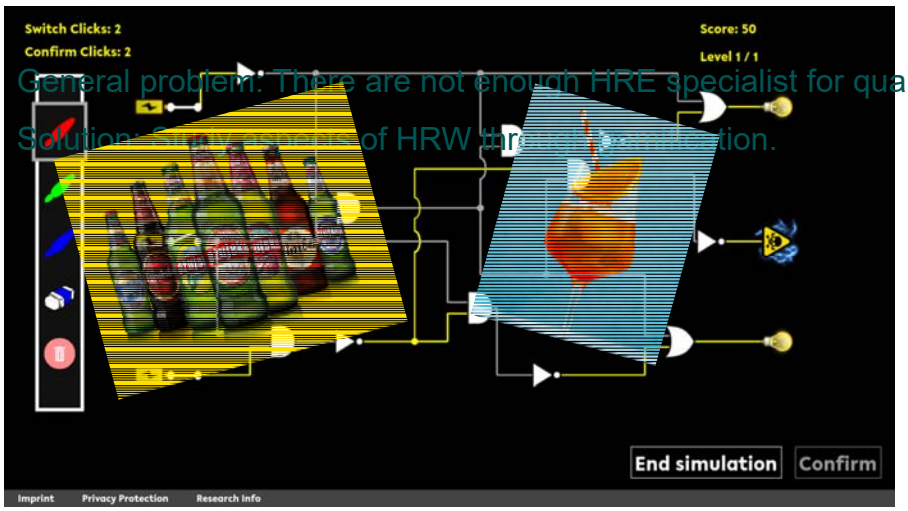
Conclusion

- First step towards a **better understanding** of cognitive processes in HRE
- **Three-phase model**
- **Working memory** might influence HRE (… but this is an **innate** ability)
- Further Research 1:
  - Can HRE be **taught** or do we just have to **pick the "right" people**?
- Further Research 2:
  - **Cognitive Obfuscation**: What are optimum ways to obfuscate circuits?
- more info:

  Becker, Wiesen, Albartus, Rummel, P: *An Exploratory Study of Hardware Reverse Engineering – Technical and Cognitive Processes*. SOUPS 2020

Ongoing research: Gamification!

- General problem: There are not enough HRE specialist for quantitative studies
- Solution: Survey aspects of HRW through gamification.



Game Dream Team:

Markus & René

- Solve simple …. not-so-simple Netlists with gates.

Vouchers for free drinks!

IF YOU LIKE OUR RESEARCH …

We are always looking for

- visiting scientists
- interns
- PhD Students
- postdocs

THANK YOU VERY MUCH FOR YOUR ATTENTION!

Christof Paar

Max Planck Institute for Security and Privacy