Hardware Security Fundamentals

98-154/18-224/18-624: Intro to Open-Source Chip Design

Why Care About Hardware Security?

- 2010 Stuxnet worm brought widespread attention to embedded attacks
  - Iranian nuclear enrichment centrifuges destroyed
- Embedded attacks becoming widespread
  - IoT botnets, Mirai, etc.
  - Attacks on vehicle rolling-codes, bank-cards, etc.
  - DEFCON Car Hacking Village, Embedded CTF, Q3K GDS CTF

Why Care About Hardware Security?

Why Should *You* Care?

- Everyone here will go on to work on very different things
  - Silicon design
  - Computer architecture
  - Embedded devices
  - System software
  - User-facing software
- What do all of these have in common?

Why Should *You* Care?

- What do all of these have in common? **They all have some kind of implication on security.**
- "Security Mindset" is very useful when building anything
  - "Security Mindset" = Always thinking about *why* a system works and how you could break it
  - If you internalize this, you'll end up building more-secure software/hardware/chips/etc.
- This lecture walks the line between embedded and hardware security, but the general concepts are very widely applicable

Fundamentals of Systems Security

- **Systems Security**: Denying a specified set of attacks from a specified set of adversaries *on our system*
- *Example:* System is this presentation, on this laptop. Adversary's goal is to prevent this lecture from happening, we want to deny this.

Fundamentals of Systems Security

- **Adversary Model**: What adversaries does our system face? What capabilities do our adversaries have? What attacks to we want to deny?
- Adversary has exploited the trackpad driver and has the ability to run arbitrary code on the computer every time the trackpad is touched.
- *Despite this, can we still prevent the adversary from compromising this lecture?*

Fundamentals of Systems Security

- **Attack Denial**: Even if the adversary has some level of compromise, can we deny further attacks on the system?
- *Can we still prevent the adversary from compromising this lecture?*
- Yes! Human defense: tell presenter to not touch the trackpad
- Yes! Technology defense: put cardboard over the trackpad

Fundamentals of Systems Security

- **Defense-in-Depth**: Multiple layers of protection against the same attack, to reduce the chance of a successful attack.
- *Can we still prevent the adversary from compromising this lecture?*
- Yes! Human defense: tell presenter to not touch the trackpad
- Yes! Technology defense: put cardboard over the trackpad
- **Doing both = defense in depth**

Fundamentals of Systems Security

- **Trusted Computing Base (TCB)**: The *underlying* layers of the system, which we assume to perform correctly, even when under attack
- In this scenario, we are assuming that the underlying system works correctly, despite the adversary:
  - CPU, RAM, Electronics
  - Bootloader, Firmware
  - Operating System

Trusted Computing Base

- Early questioning of a TCB in Ken Thompson's Reflections on Trusting Trust
- Main idea: what if a compiler contains self-replicating malware that is inserted into every new compiler that it is used to compile, silently propagating into everything
- "No amount of source-level verification or scrutiny will protect you from using untrusted code"
- Lawrence Kesteloot's short story Coding Machines is a more sci-fi take on the same concept

Why are Systems Insecure?

Why are Systems Insecure?

- **(1)** System does not implement intended logic (aka “bugs”)

```
1  char name[16];
2  bool is_admin;
3
4  printf("What is your name?\n");
5  fgets(name, 32, stdin); // oops
```

Attacker gives input of “AAAAAAAAAAAAAAAA”. First 16 characters go into `name` and then the last byte overwrites `is_admin` to be 01, giving the user admin access.

Why are Systems Insecure?

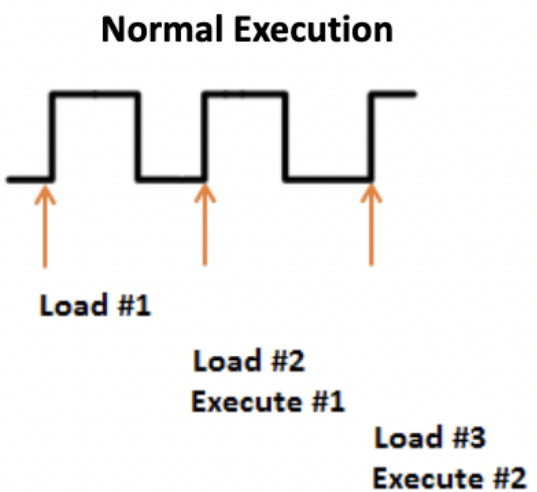
- (2) Adversary exploits user-error to attack system
  - *System Misconfiguration:* System uses a password to protect against unauthorized access, but the user sets their password to “12345”
  - *User Exploitation:* Attacker sends the user a phishing email to trick them into entering their password onto a fake website
- Main idea: adversary can attack anywhere in the system, all the way from the user-level (PEBCAK) to the hardware level

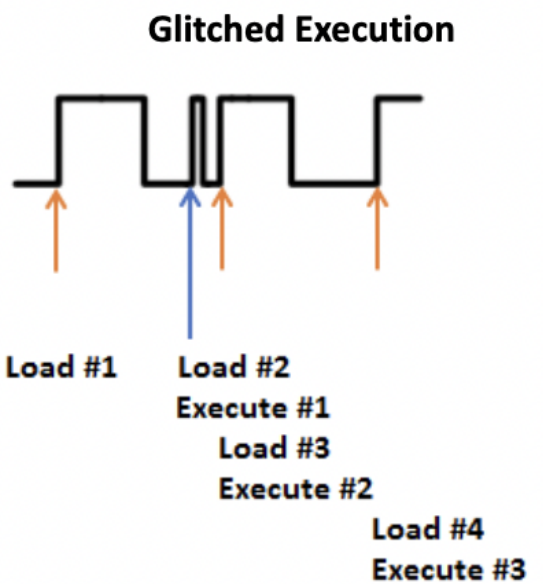
Why are Systems Insecure?

- (3) System is designed poorly but implemented correctly
  - System allows updating firmware and verifies the firmware using a hash function to protect against corruption
  - Attacker creates a malicious firmware and computes its hash, allowing them to install it onto the system
  - Should have instead used asymmetric cryptography to sign the firmware using a private key, which the attacker wouldn’t be able to forge.

Why are Systems Insecure?

- (4) Compromise of TCB: “Trusted” != “Trustworthy”
  - We assume CPU will execute every given instruction
  - Attacker glitches clock signal to cause the hardware to skip an instruction, bypassing a security check.





Why are Systems Insecure?

- **(5)** Adversary exploits “side channels” to extract data from the system
  - Side Channel = some method by which an attacker can obtain information from a system in an unintended manner
  - For example, the power usage of a processor is correlated with the data being processed. In some cryptographic algorithms, this can be used to leak information about the secret-key.

Why are Systems Insecure?

- (6) Incomplete adversary model
  - Bit of a cat-and-mouse game: it seems impossible to get a complete list of adversary capabilities
  - Part of DARPA's mission statement is "preventing strategic surprise"

Why are Systems Insecure?

- (7) Adversary model changes over time
  - Designers of old aircraft systems did not expect passenger access to internal network, but then entertainment systems and in-flight Wi-Fi were retrofitted, increasing attack surface

Challenges in Embedded Security

1. Limited resources (compute, memory, power)
2. Difficult to patch if deployed in the field
3. May be prone to physical tampering by attackers
4. Specialized peripherals, low-level system design
5. Can rarely assume a TCB, must worry about security from the transistor-level up to the user-level

Attacks: Side-Channel

Side-Channel Basics

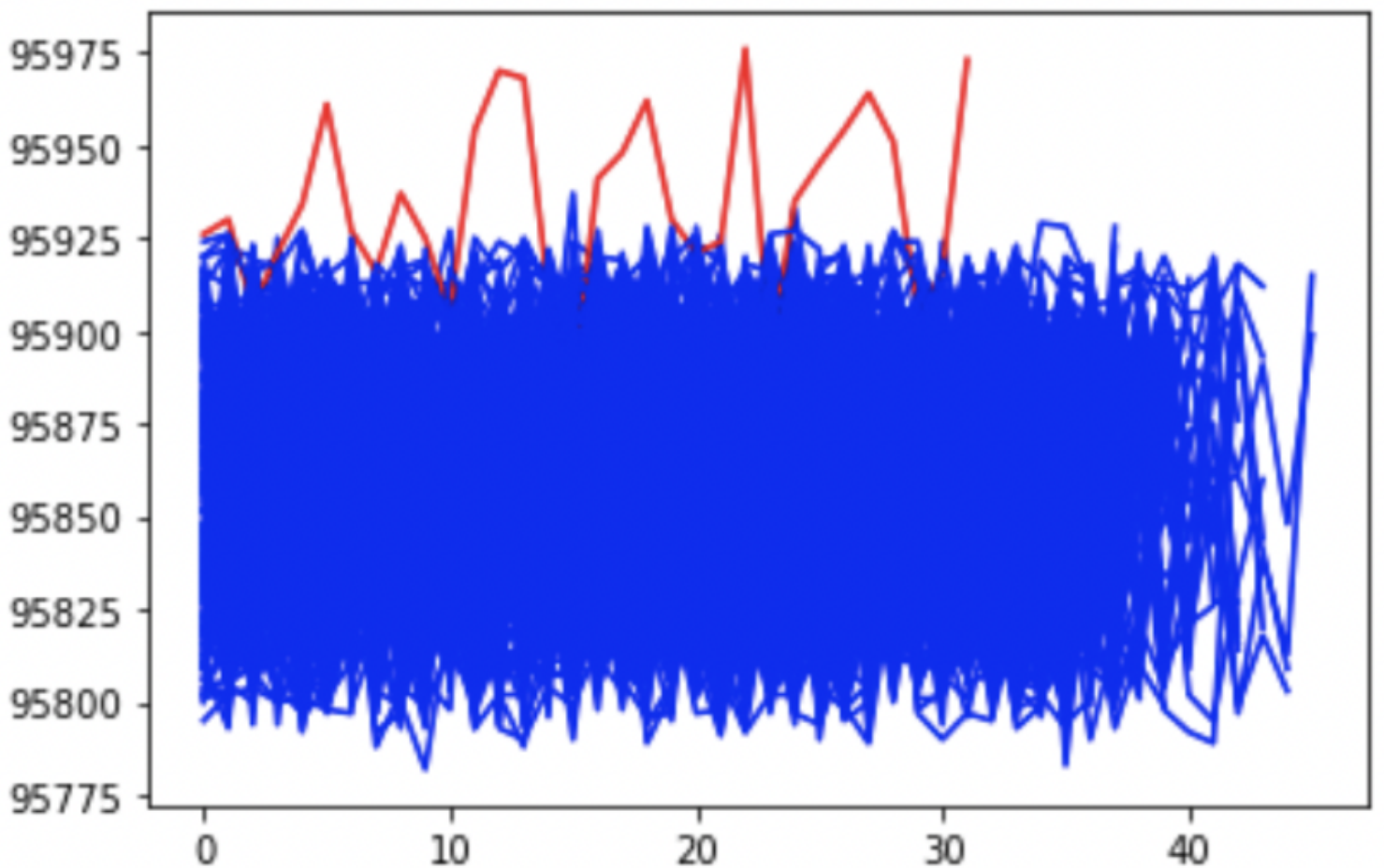
- **Side-channel**: Leaks information about a system in an unintended manner
  - Timing
  - Acoustic
  - Power-draw
  - Electromagnetic (TEMPEST)
  - Cache-based (Meltdown)

Side-Channel Basics

```
for (int i = 0; i < 16; i++) {
    if (user_input[i] != password[i]) {
        return false;
    }
}
return true;
```

```
bool is_correct = true;
for (int i = 0; i < 16; i++) {
    if (user_input[i] != password[i]) {
        is_correct = false;
    }
}
return is_correct;
```

Dealing with Noise



Differential Power Analysis

- Cryptographic functions like AES256 are a known function of the encryption-key and inputs to determine the outputs
- First few steps of AES encryption:
  - Take 16 bytes of input data
  - XOR it with the encryption key
  - Run it through a substitution cipher
  - Permute it in a specific way
  - (repeat for 9-13 rounds depending on AES variant)

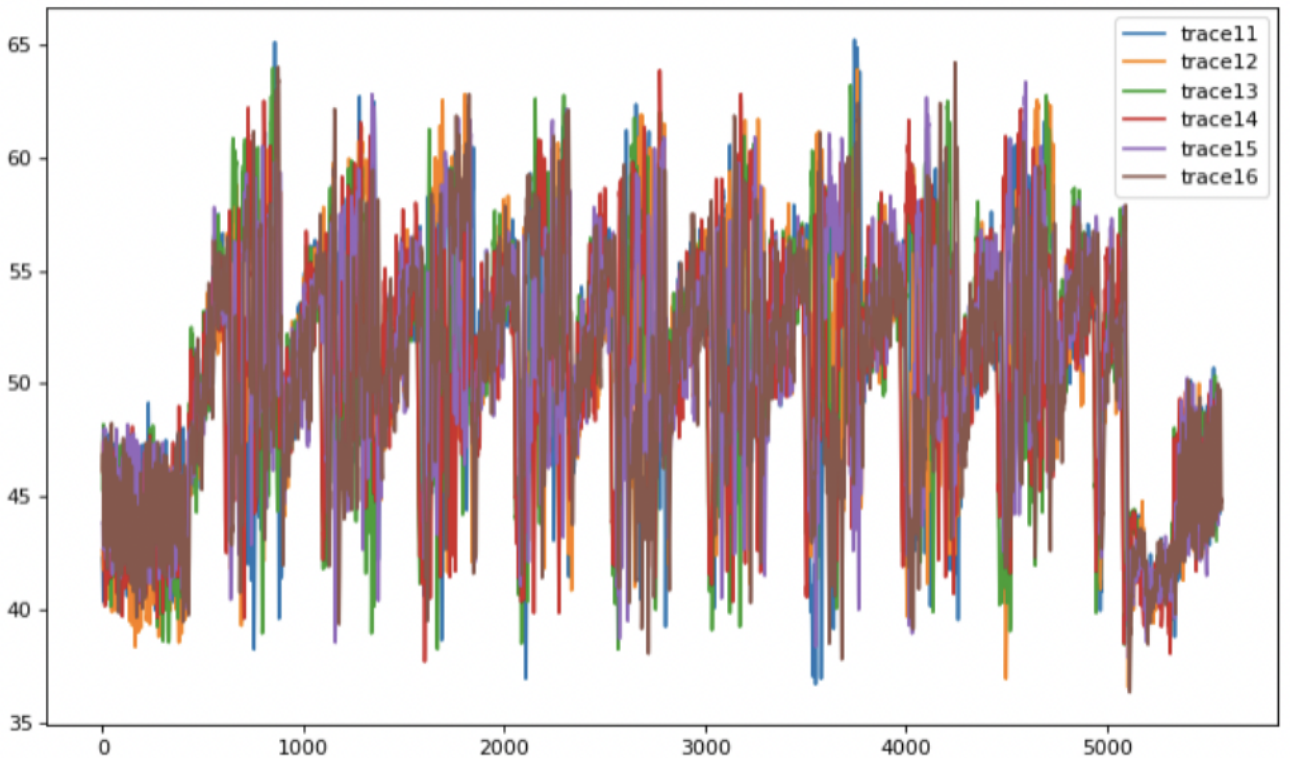
Differential Power Analysis

- Effectively, first round computes
  - `out[i] = SBox(input[i] XOR key[i])`
- Power consumption of a chip is correlated with the Hamming-weight (number of 1 vs 0 bits) of the data being processed
- Attacker can use known input data and correlate with power-consumption to derive key

Differential Power Analysis

1. Input various random bytes for `input[0]`, and measure the power consumption
2. For every possible (256 possibilities) byte `key[0]` and every inputted byte `input[0]`, compute `Hamming(SBox(input[0] XOR key[0]))`
3. For each possible byte `key[0]`, correlate the Hamming weight with the power-consumption for every inputted byte.
4. One possible value should have a significantly higher correlation; that byte is the actual value of `key[0]`
5. Repeat for the remaining bytes of the input and key (1..15)

Differential Power Analysis



0	1	2
18 0.453	a0 0.560	28 0.402
47 0.292	6c 0.271	8c 0.273
ca 0.258	4b 0.259	b0 0.249
13 0.250	08 0.257	f8 0.248
1e 0.246	67 0.252	2d 0.248

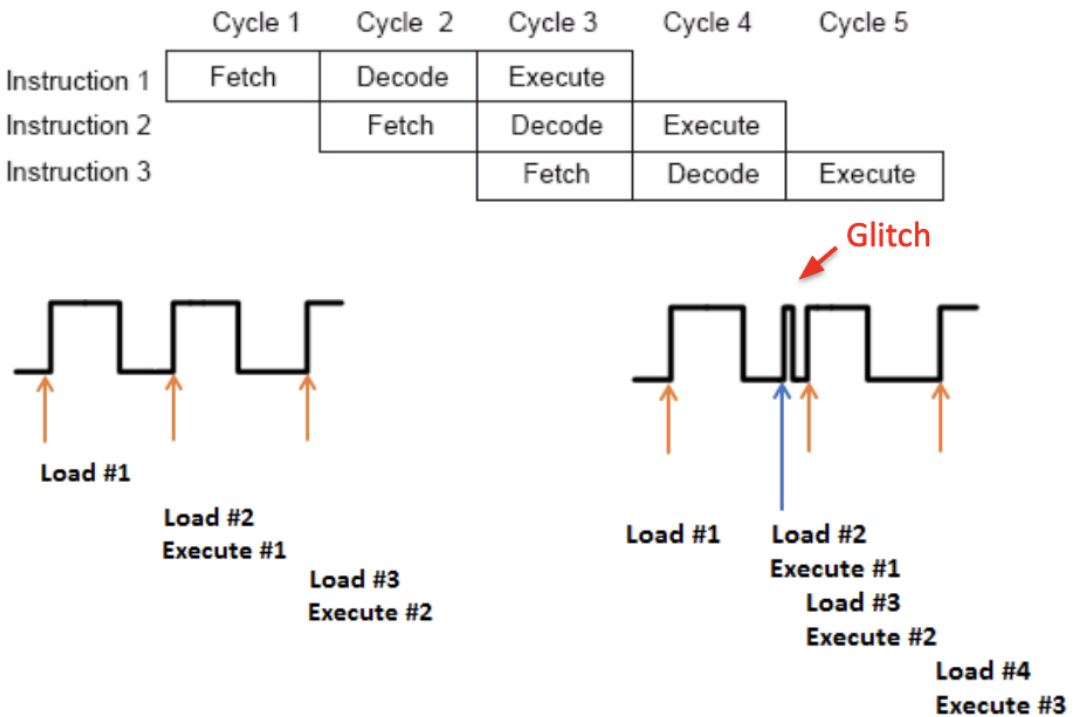
Attacks: Fault Injection

Fault Injection

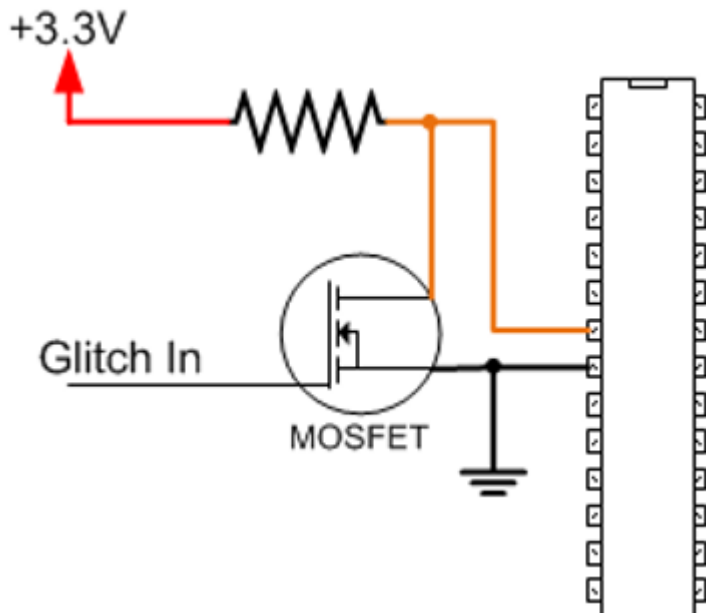
```
int main(){
     bool password_check;

     password_check = some_fancy_checking_function();
     if( password_check == FAILED ) halt_and_catch_fire();

    // Allow user to access secret data

}
```

What if we could just skip the loading or checking instructions?

Fault Injection by Clock-Glitching

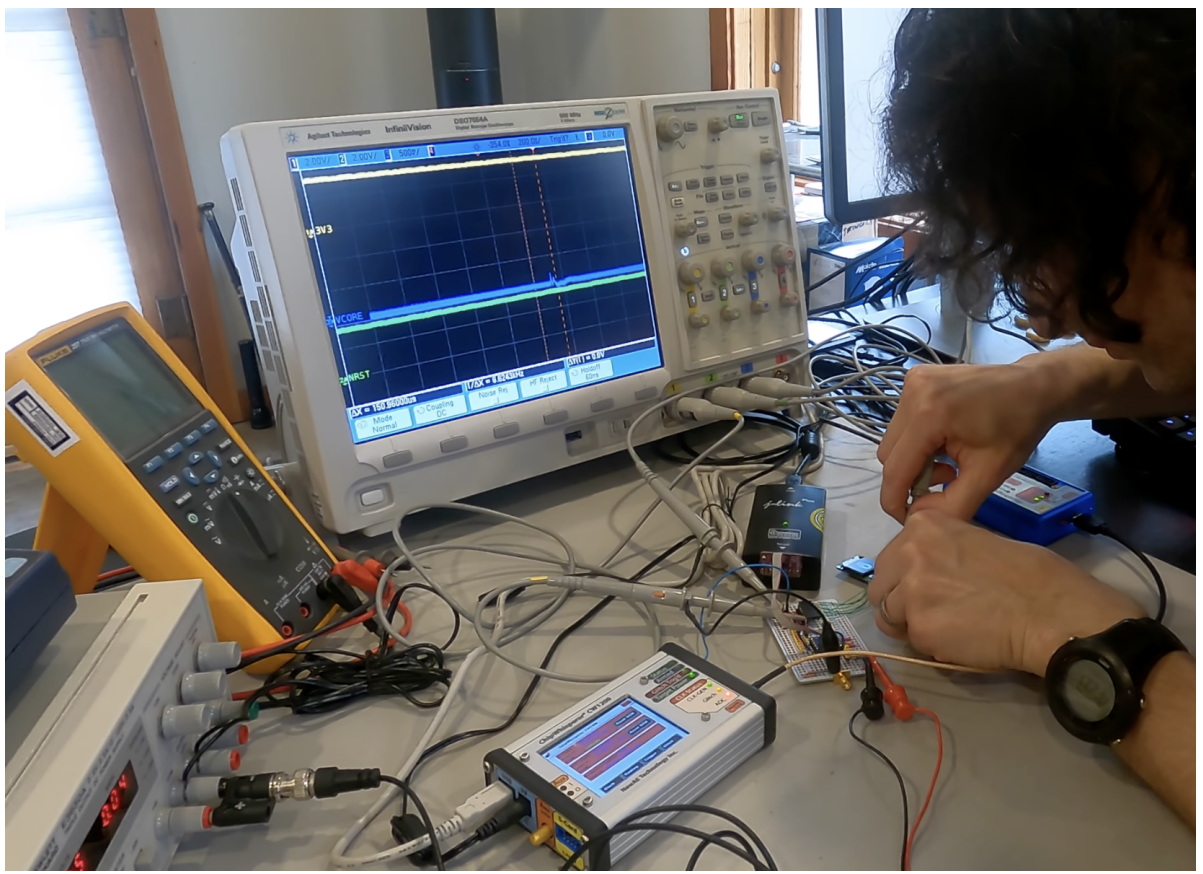


Fault Injection by Power-Glitching



Example: Trezor Wallet

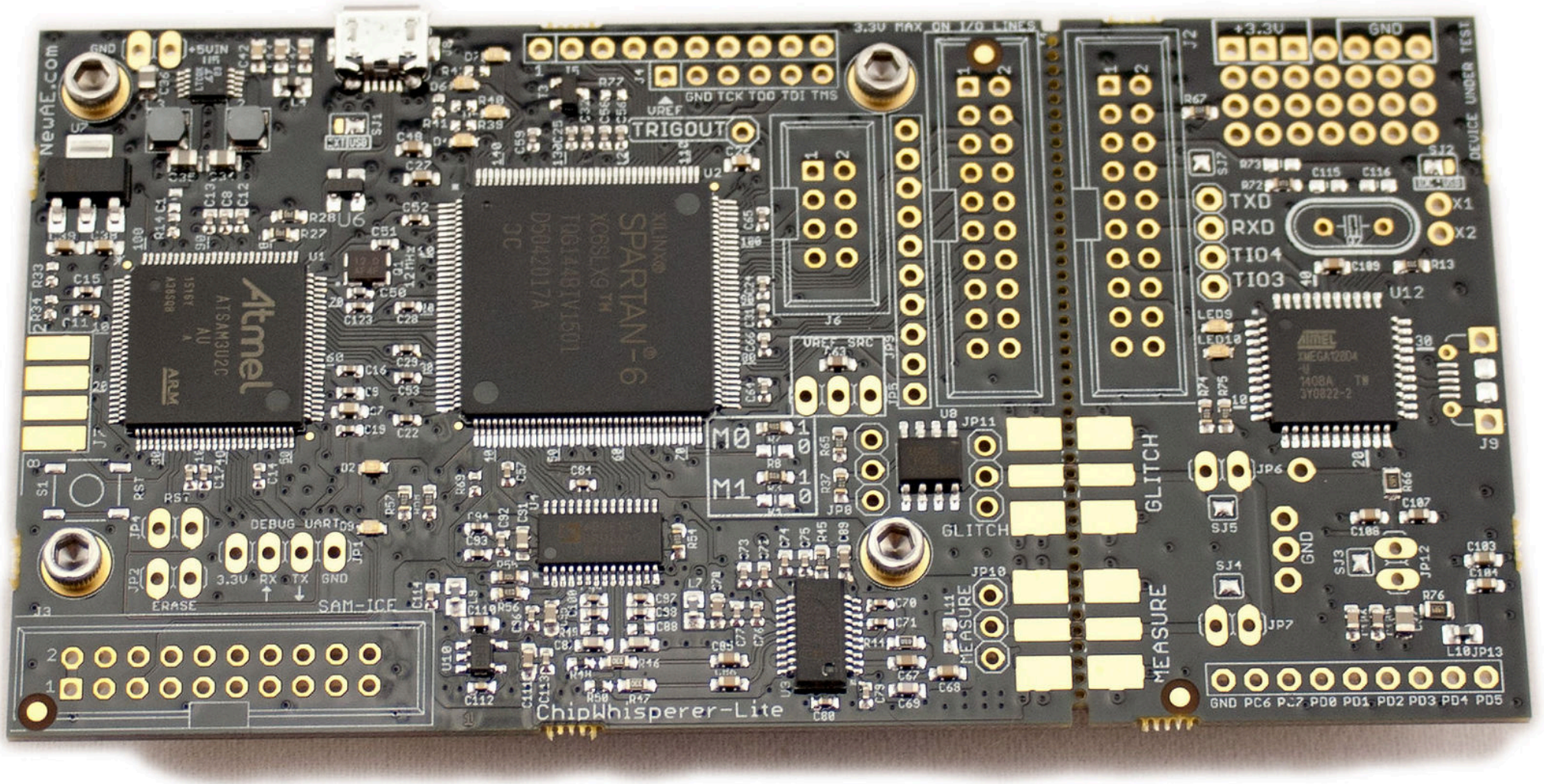
Security researcher Joe Grand used fault-injection to recover $2M from a hardware crypto-wallet with a forgotten password (Fascinating video on how it was done)



Practical Hardware Attacks

- ChipWhisperer Lite is a low-cost, practical platform for developing hardware-based attacks
  - Even used in the research literature
- 100 MSPS power capture for side-channels
- Sub-nanosecond clock glitching
- Crowbar-based voltage glitching

Practical Hardware Attacks



Practical Hardware Attacks

- Hardware Hacking Handbook: an approachable introduction to embedded attacks
  - Goes into a *lot* of depth on how to attack (and defend) embedded systems
  - Free through CMU through O'Reilly
- ChipWhisperer-Jupyter
  - Hands-on tutorials on using ChipWhisperer for fault-injection and side-channel attacks

Attacks: Supply-Chain

Supply-Chain Attacks

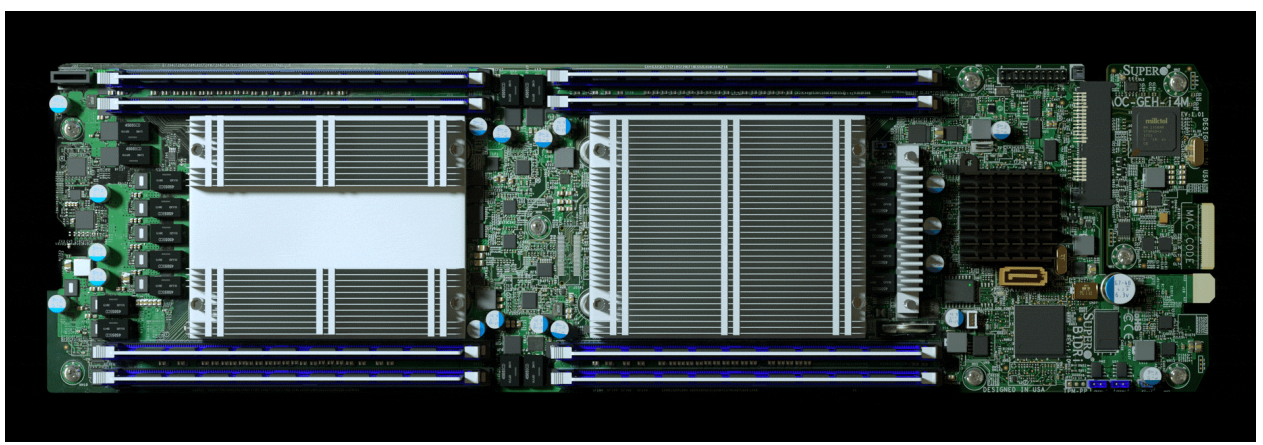
- Supply-chain comprises anything below the level which you are working on. Such attacks can be hard-to-detect.
- Software example:
  - Let LibX be a very popular Python library
  - The library's developer's machine gets hacked and used to publish an "update" to LibX containing malware.
  - LibX users pick up this update, suddenly the malware is running on millions of machines around the world.
- Real-world example: the 2016 `left-pad` incident

Supply-Chain Attacks on Hardware

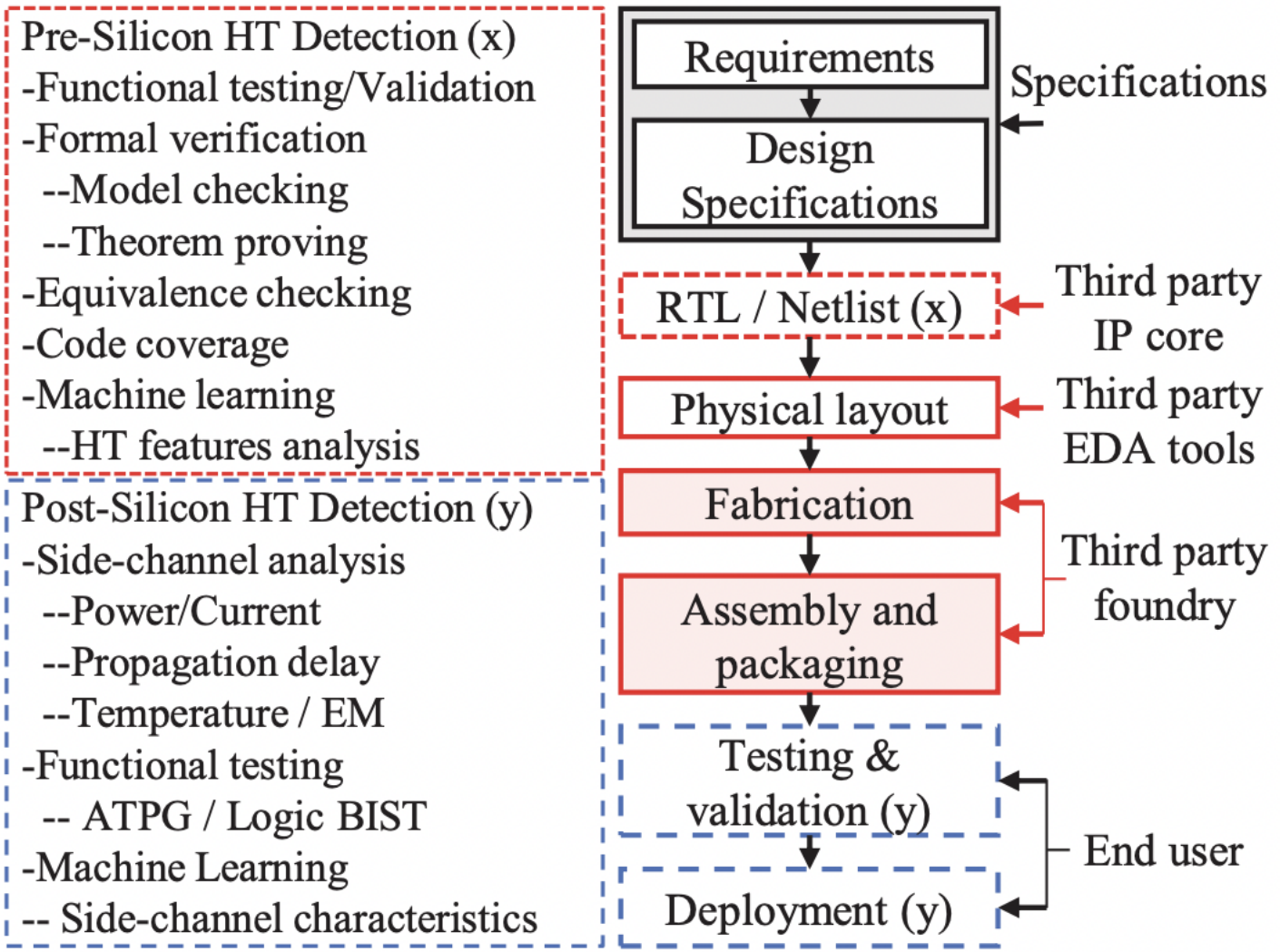
- Nation-state-level actor could compel chipmaker to insert malicious logic at the foundry
- Use of special trigger condition to avoid detection, but still able to be triggered by attacker
- Very, very difficult to truly protect against (fortunately, most of us don’t need to worry)
- This is part of why chips used in defense are manufactured domestically (SkyWater, GlobalFoundries, etc. in the US)

Supply-Chain Attacks on Boards

- In 2018, Bloomberg reported on a supposed board-level attack by a Chinese army unit on Supermicro, a manufacturer of server motherboards
- Extremely small chips, disguised as passive analog components, were discovered on Supermicro motherboards that were not in the original designs



Hardware Trojan Attack Surface



Open-source EDA (somewhat) reduces attack surface

Hardware Trojan Detection

- Logic-based Detection
  - Use automated test-pattern generation to attempt to trigger the hardware-trojan and cause unexpected behavior (inefficient for large chips)
- Power-analysis Detection
  - Added gates from a hardware-trojan increase leakage and/or dynamic power consumption
  - Possible to design trojans without a noticable increase in power or # of gates

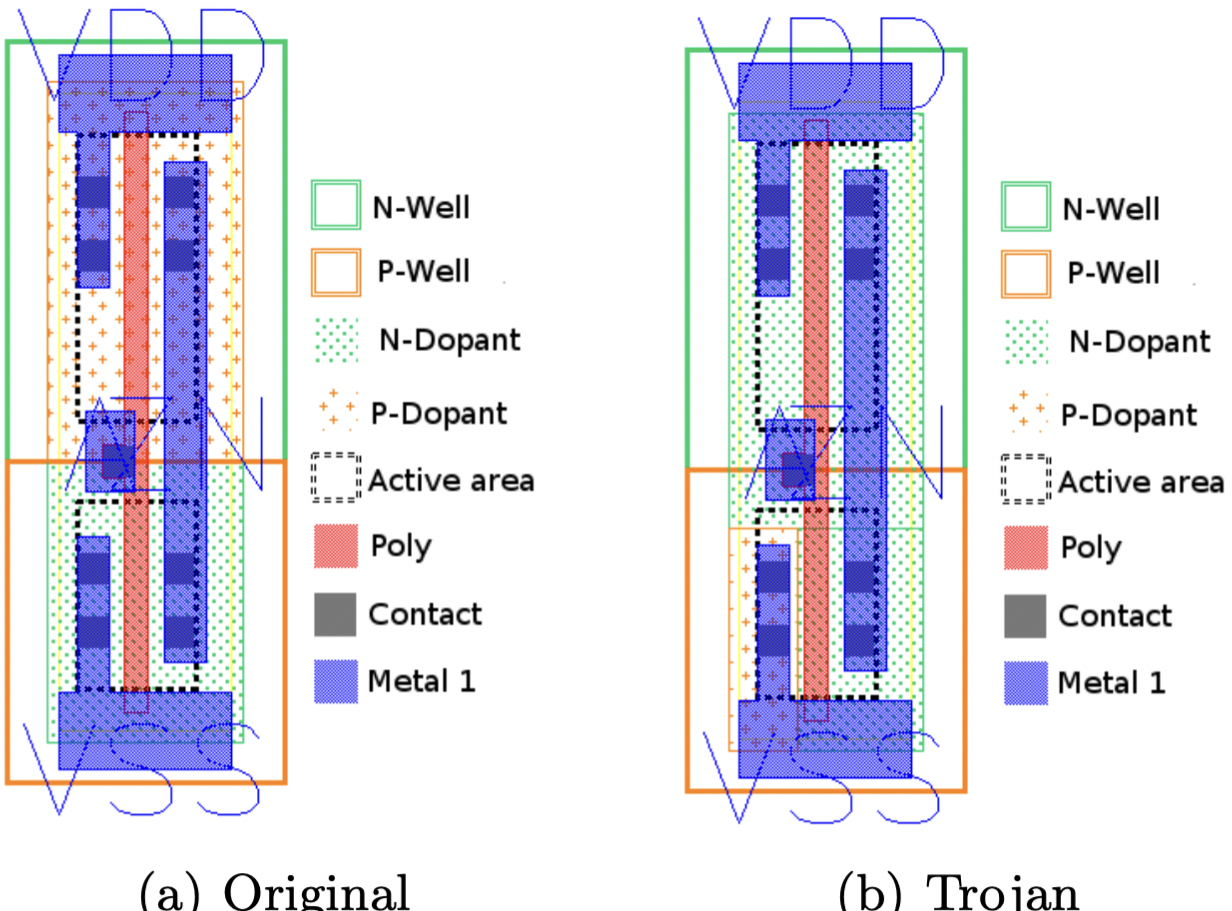
Hardware Trojan Detection

- Optical Detection
  - Delayer and scan the chip, visually compare it against design files
  - Inefficient, error-prone process to scan an entire chip
  - Can’t detect features such as dopant-polarity

Dopant-Based Hardware Trojans

- Optical scanning is often used as a form of reverse-engineering. As such, obfuscation methods have been developed (and even used in real hardware) which vary the dopant polarity of standard-cells to prevent reverse-engineering.
- This works because optical methods are not able to distinguish dopant polarity, only metal layers
- In 2013, Becker et al. showed how the same method could be used to develop undetectable hardware trojans.

Hardware Trojans: Inverter



(a) Original (b) Trojan

Fig. 1. Figure of an unmodified inverter gate (a) and of a Trojan inverter gate with a constant output of V_{DD} (b).

Hardware Trojans: Flip-Flop

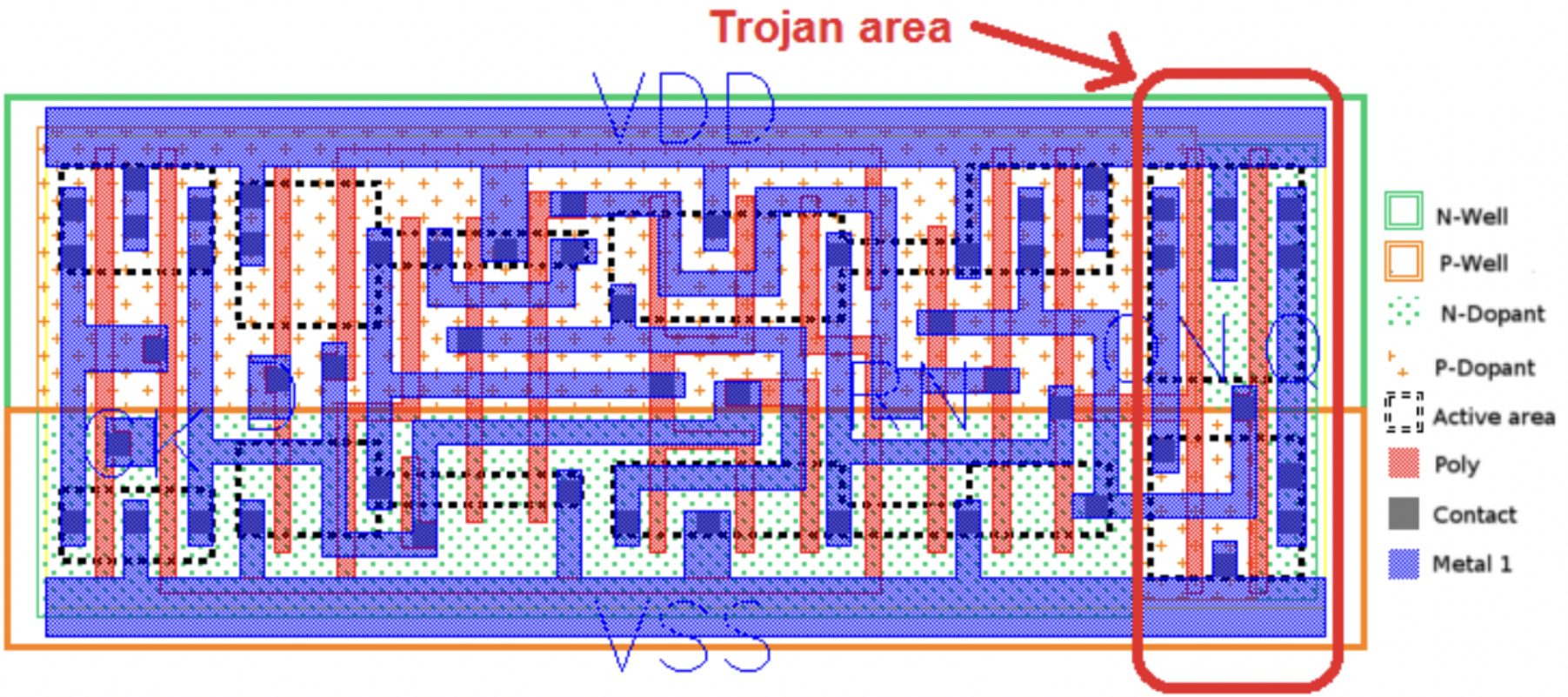


Fig. 2. Layout of the Trojan DFFR_X1 gate. The gate is only modified in the highlighted area by changing the dopant mask. The resulting Trojan gate has an output of $Q = V_{DD}$ and $QN = GND$.

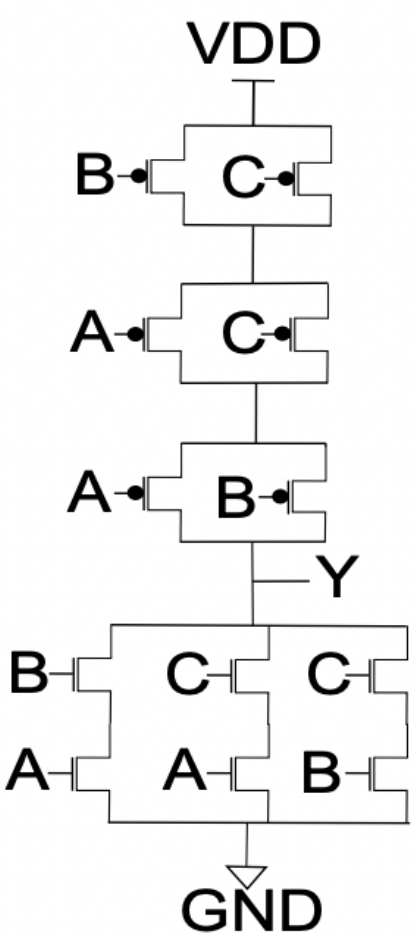
Hardware Trojans: RNG

- Becker et al. compromise Intel's RNG by fixing 96 of the 128 bits in the entropy register, leaving only 32 bits of entropy
- Entropy register is tested with 32-bit built-in self-test (BIST), which can be rigged
- By rigging the entropy register, output still "appears" random, so it passes NIST and FIPS statistical tests

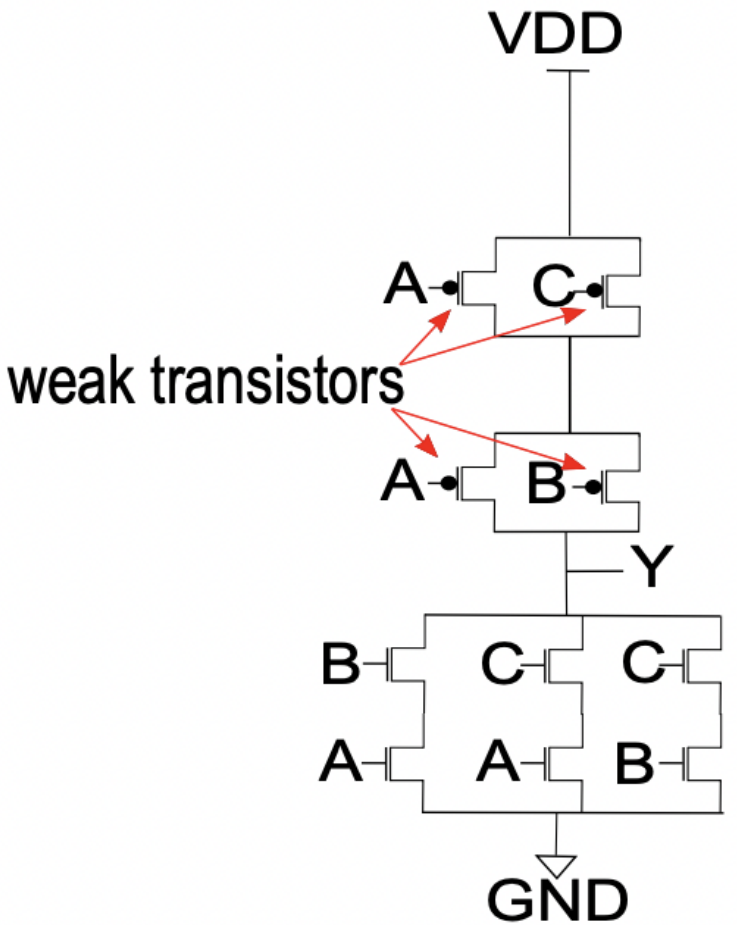
Hardware Trojans: Side-Channel

- Leak data-bits by forcing a stronger-than-normal power-correlation to those bits
- Changing drive-strength of specific transistors in a very-small number of gates can be enough to leak secret information
- Undetectable by any amount of functional testing, since the logic function being computed hasn’t changed at all

Hardware Trojans: Side-Channel

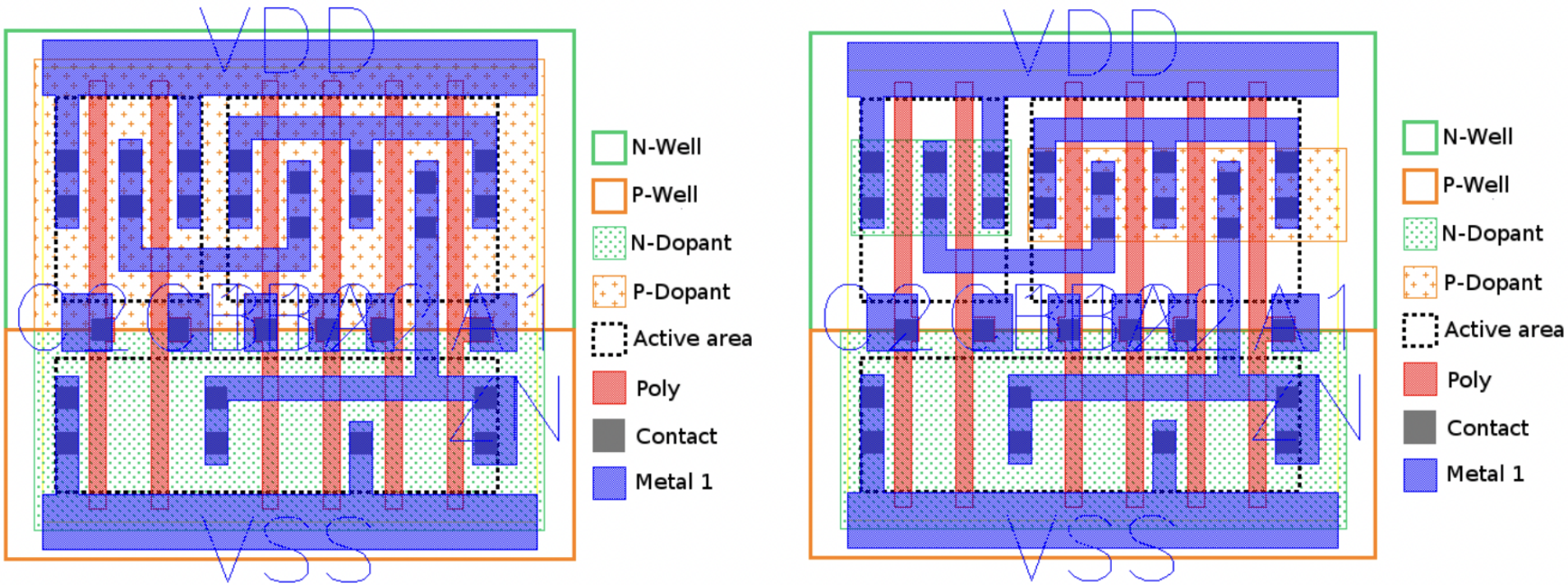


a) Trojan free AOI222 Gate



b) Trojan AOI222 Gate

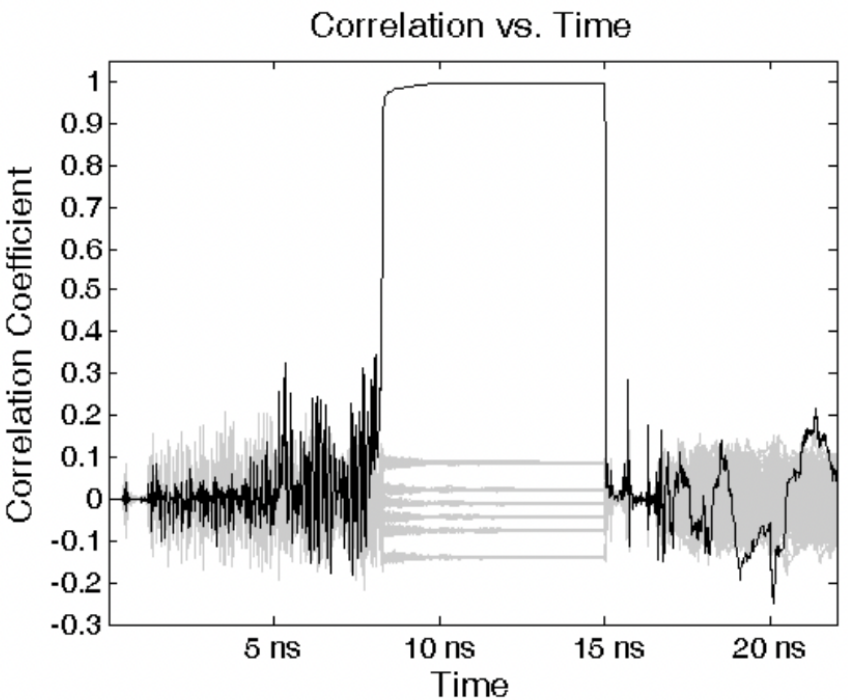
Hardware Trojans: Side-Channel



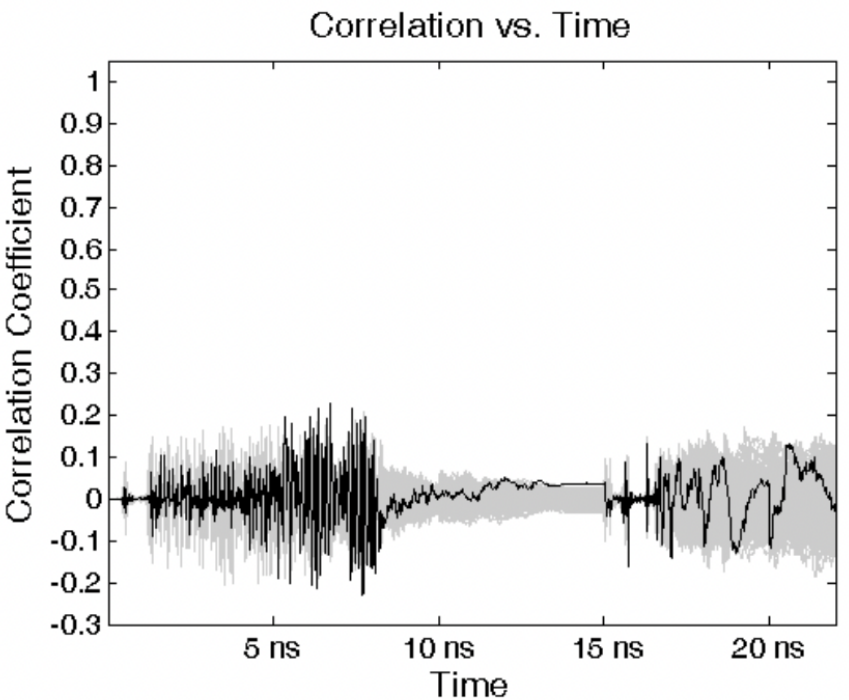
(a) Trojan-free AOI gate

(b) Trojan AOI gate

Hardware Trojans: Side-Channel



(a) Trojan design



(b) Trojan-free design

Hardware Trojan Defenses: FPGAs

- Building a design on top of an FPGA improves security because the foundry wouldn't know *which* gates or flip-flops to backdoor
- Place-and-route is inherently heuristic, so every time the same design is compiled, it would be laid out differently onto the FPGA
- Backdooring every logic block in the FPGA would be much easier to detect

Hardware Trojan Defenses: FPGAs

- Precursor, by bunniestudios, is a mobile device with a fully open-source design, based around an FPGA as its main CPU (full auditability of every aspect of the hardware, CPU, and software)

Hardware Trojan Defenses: FPGAs

- Going one level further - Gabriel Somlo of CMU SEI worked on developing a self-hosting CPU on an FPGA, which could replicate its own bitstream
- This ensures that the bitstream is not vulnerable to malware on the machine being used to run the FPGA toolchains
- Makes “Trusting Trust”-style attacks magnitudes harder to execute
- Gabriel Somlo’s website

Hardware Trojan Defenses: eFPGAs

- Mohan et al. (2021) of CMU ECE developed a framework for tightly-coupled design of embedded FPGA fabrics which can be used to protect critical portions of designs
- For example, a CPU could include an attached eFPGA which would then be loaded (by the end-user) with the bitstream for a cryptographic accelerator
- This would protect the design against reverse-engineering (since the accelerator wouldn't be loaded until after manufacturing) and tampering (since the layout of the accelerator would not be known at fabrication-time)

Shared Tenancy Attacks

- Cloud providers such as AWS provide large FPGAs for on-demand usage
- FPGAs are shared between tenants (sometimes multiple tenants at once, sometimes one-after-another)
- Raises concerns about data-leakage and other attacks

Shared Tenancy Attacks: Side-Channels

- Possible to build logic on an FPGA (ring-oscillator, time-to-digital sensors) which can detect tiny voltage fluctuations in the supply-voltage
- This has been shown to be sufficient to carry out power-based side-channel attacks on AES encryption running elsewhere on an FPGA
- Similar attacks can be executed against the built-in CPU core of the FPGA

Shared Tenancy Attacks: Fault-Injection

- Voltage-glitching attacks possible by abusing clock-gating logic to place unreasonable load onto the FPGA, causing momentary glitches in power
- Similar attacks possible using ring-oscillators, which can run at very-high switching frequencies using only a small amount of logic
- RowHammer attacks can potentially be used to corrupt bits in other cores' DRAM

Shared Tenancy Attacks: Covert Channels

- Sometimes possible to add backdoors to an IP core which is then used in a design, but can't exfiltrate data if the IP core doesn't have any external interfaces
- Voltage-based and thermal-based attacks can be used to leak small amounts of data to other logic on the same FPGA, logic installed *later* onto the same FPGA, and sometimes even logic on other FPGAs.

Taxonomy of Hardware Attacks (Papp et al, 2015)