Finding and Mitigating Side-channel Leakage in Embedded Architectures

Patrick Schaumont (PI), William Diehl (Co PI)
Virginia Tech, ECE Department


The Problem

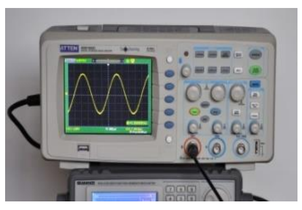
Embedded devices vulnerable to Side Channel Attacks (SCA)

- Mathematically secure but physically breakable

Generalized solution to compose SCA-resistant SW remains fleeting

- Languages, compilers, ISA, and Hardware constantly changing
- Automation remains elusive



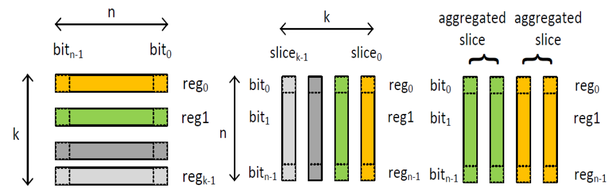


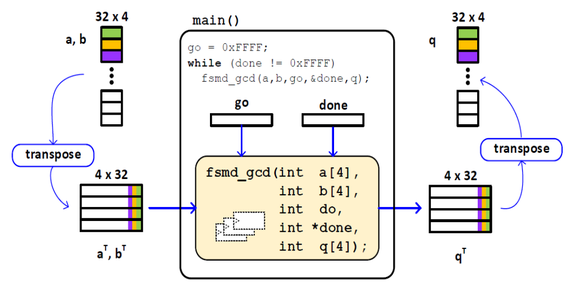

http://www.pd4pic.com/satellite-flat-icon-cartoon-dish-antenna-signal.html

Our Objectives

Meet above challenges head-on through *composable*, *multi-faceted*, *automated* approach

- Restructure software into *bitsliced code* to facilitate *automated countermeasure insertion*
- *Automated flow* to *find*, *evaluate* mode of leakage, and apply *mitigation* techniques





Key Challenges

Generation of bitsliced code

- Control intensive processes are challenging

Countermeasure insertion is "hit & miss"

- Most embedded designers are not experts in security

Countermeasure effectiveness unpredictable

- Many layers of abstraction (HLL, compilers, ISA, logic gates)

Verification is time-consuming

- Difficult in HW; How to get accurate results in simulation?

Significance

Inhibits wide-spread exploration of security & efficiency

Countermeasures either not, or incorrectly implemented, in embedded architectures

Mathematically "secure" countermeasures end up not being secure

Verification & mitigation of leakage not performed due to time and expense

Scientific Impact

Generalized approach to generate bitsliced code

- Improved compiler design
- Improvements in High Level Synthesis (HLS)
- On-the-fly adjustment of security and variable precision

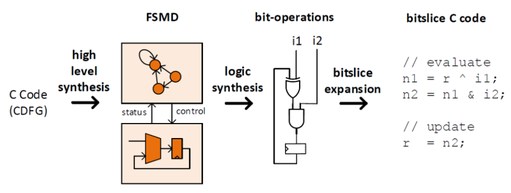
Improved understanding of side-channel "theory versus reality"

- Better understanding of linkage between compiled language and ISA
- Close the gap between mathematically-derived randomness requirements, and empirical observations

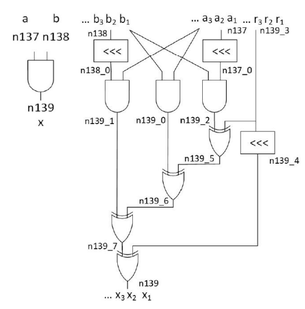
Automated insertion of countermeasures; identification and mitigation of leakage

- Secure design depending on arcane science is not secure
- Design flows available to designers as part of main-stream EDA tools

Generate Bitsliced Code

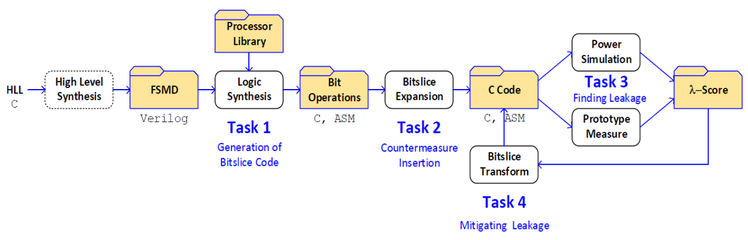


Insert Countermeasures

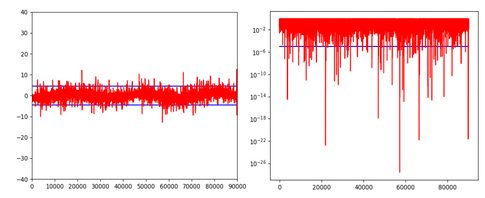



```
n137_0 = rot(n137);
n138_0 = rot(n138);
n139_1 = n138_0 & n137;
n139_0 = n138 & n137;
n139_2 = n137_0 & n138;
n139_3 = sys(r);
n139_4 = rot(n139_3);
n139_5 = n139_2 ^ n139_3;
n139_6 = n139_5 ^ n139_0;
n139_7 = n139_6 ^ n139_1;
n139 = n139_7 ^ n139_4;
```

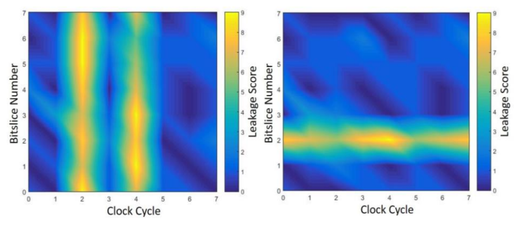
Solution: ***Automated Flow*** to ***Generate Bitsliced Code***, ***Insert Countermeasures***, ***Find***, and ***Mitigate Leakage***



Find Leakage

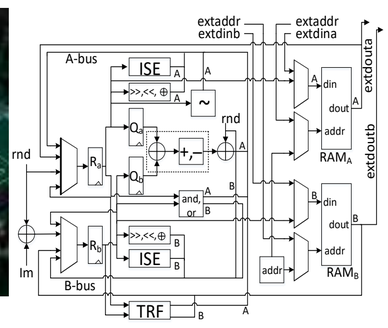


Mitigate Leakage



Broader Impacts

http://www.linuxgizmos.com , http://www.element14.com



http://www.vectorstock.com, http://businessinsider.com, http://www.grid.org

Courses

- ECE 4530 Hardware Software Codesign
- ECE 5580 Cryptographic Engineering
- ECE 5520 Secure Hardware Design

References

P. Kiaei, D. Mercadier, PE. Dagand, K. Heydemann, P. Schaumont, "SKIVA: Flexible and Modular Side-channel and Fault Countermeasures," IACR ePrint 2019/756

W. Diehl, A. Abdulgadir, J. P. Kaps, "Vulnerability Analysis of a Soft Core Processor through Fine-grain Power Profiling," IACR ePrint 2019/742