How to Protect RISC-V Against Side-channel Attacks?

Elke De Mulder
Samatha Gummalla
Michael Hutter
Introduction

Intuitively:
- $\neq$ power consumption
  $\rightarrow$ Open/closed transistor
  $\rightarrow$ Bit Flip Register Yes/No
  $\rightarrow$ $\neq$ values
- Intermediate values = $f$(secret)

Side Channel Analysis == use of measurable, physical properties of a chip during the execution of an operation to extract secret information used within the chip
Countermeasures

- State-of-the-art: Boolean masking

- E.g. value $x$ represented as a tuple $(x_0, x_1)$
  \[
  x = x_0 \oplus x_1
  \]
  with $x_0$ random

- Algorithms are adjusted to work with this representation such that each intermediate is statistically independent of $x$ and $y$
Side-channel Related Software Leaks

Where are these secret related values used in a SW implementation on a processor?

- ALU
- Register bank
Side-channel Related Software Leaks

Where are these secret related values used in a SW implementation on a processor?

- ALU
- Register bank
- Memory and memory interface
Side-channel Related Software Leaks

Where are these secret related values used in a SW implementation on a processor?

- ALU
- Register bank
- Memory and memory interface
- ALU input/output registers
Side-channel Related Software Leaks

Where are these secret related values used in a SW implementation on a processor?

- ALU
- Register bank
- Memory and memory interface
- ALU input/output registers
- Forwarding
Side-channel Related Software Leaks

Where are these secret related values used in a SW implementation on a processor?

- ALU
- Register bank
- Memory and memory interface
- ALU input/output registers
- Forwarding
- Cache
Side-channel Related Software Leaks

Where are these secret related values used in a SW implementation on a processor?

- ALU
- Register bank
- Memory and memory interface
- ALU input/output registers
- Forwarding
- Cache
- Branch Prediction Unit
Side-channel Related Software Leaks

Where are these secret related values used in a SW implementation on a processor?

- ALU
- Register bank
- Memory and memory interface
- ALU input/output registers
- Forwarding
- Cache
- Branch Prediction Unit
- Fetch and Decode Unit
Side-channel Related Software Leaks

Where are these secret related values used in a SW implementation on a processor?

- ALU
- Register bank
- Memory and memory interface
- ALU input/output registers
- Forwarding
- Cache
- Branch Prediction Unit
- Fetch and Decode Unit
- JTAG/Debug interface
Side-channel Related Software Leaks

Where are these secret related values used in a SW implementation on a processor?

- ALU
- Register bank
- Memory and memory interface
- ALU input/output registers
- Forwarding
- Cache
- Branch Prediction Unit
- Fetch and Decode Unit
- JTAG/Debug interface
- ....
Side-channel Related Software Leaks

How does power related information manifest itself inside these components?

- **Direct values**
  Register writing, values going through the ALU, values stored in micro-architecture registers (e.g. forwarding register, register at the entrance or exit of an ALU, ...)

- **Data-overwrite values**
  Overwriting a value with a different value exhibits a power consumption usage related to the bit difference between the two values

- **At the circuit-level through glitches, wire interconnects, etc.**
Countermeasure Example: AND Operation

1. Generate a random value $d$
2. $y_0 = y_1 = d$
3. $t = a_0 \land b_0$
4. $y_1 = y_1 \oplus t$
5. $t = a_1 \land b_0$
6. $y_1 = y_1 \oplus t$
7. $t = a_0 \land b_1$
8. $y_1 = y_1 \oplus t$
9. $t = a_1 \land b_1$
10. $y_1 = y_1 \oplus t$
Countermeasure Example: AND Operation

1. Generate a random value $d$
2. $y_0 = y_1 = d$
3. $t = a_0 \land b_0$
4. $y_1 = y_1 \oplus t$
5. $t = a_1 \land b_0$
6. $y_1 = y_1 \oplus t$
7. $t = a_0 \land b_1$
8. $y_1 = y_1 \oplus t$
9. $t = a_1 \land b_1$
10. $y_1 = y_1 \oplus t$

Not statistically independent
Countermeasure Example: AND Operation

1. Generate a random value $d$
2. $y_0 = y_1 = d$
3. $t = a_0 \land b_0$
4. $y_1 = y_1 \oplus t$
5. $t = a_1 \land b_0$
6. $y_1 = y_1 \oplus t$
7. $t = a_0 \land b_1$
8. $y_1 = y_1 \oplus t$
9. $t = a_1 \land b_1$
10. $y_1 = y_1 \oplus t$

Not statistically independent
Take-away

• Creating SCA secure SW implementations on unknown HW
  == Not EASY!

• Lots of testing and lots of theoretical and practical knowledge required from a SW developer
An SCA-protected RISC-V Architecture

• Idea: “transparent hardware-protection layer”
  • Software does not need to take care about DPA or side-channel leakage
  • Software can be written in a classical “unprotected” way
• Custom RISC-V design with 5 pipeline stages
• Added features to counteract side-channel analysis
  • Random number generator (RNG) integration
  • Masking countermeasure to protect data path + register bank
  • Memory-access leakage protection
RISC-V protected

- Data from/to memory gets masked/unmasked inside the CPU boundary
- Register bank is doubled to store both mask shares
- Data path is processing 2 shares
- ALU uses state-of-the-art provable secure masking techniques to avoid leakage
SCA-protected Memory Access

• Storing both shares would require too much overhead...

• Idea: memory encryption with on-the-fly session-key calculation
  • Session keys are a function of a random seed and the memory address
  • Software can choose from at least 2 available session keys
  • Session keys can be updated via dedicated Control Status Registers (CSR)
## Side-channel Leakage Assessment

How to assess side-channel security?

- Any data dependent power consumption has the potential to reveal secrets
- *Intuitively:* Can I distinguish between power measurements of a fixed input to the algorithm and a random input? Any statistical significant difference = problematic
- Statistical significance measured by Welch’s t-test, threshold level set at $4.5\sigma$

\[
t(i) = \frac{\mu_A(i) - \mu_B(i)}{\sqrt{\frac{\sigma_A^2(t)}{N_A} + \frac{\sigma_B^2(t)}{N_B}}}
\]
Implementation and Testing
Examples of Cryptographic Implementations: AES
Examples of Cryptographic Implementations: AES

- Input Leak
- Average Power Consumption
- Output Leak
- T-Test Thresholds
Examples of Cryptographic Implementations: SHA-2
Summary

- Implemented a DPA-hardened RISC-V core

- Idea: software implementors can write unprotected code and still achieve 1st order DPA security
Thank You
Q&A