universität
innsbruck

Defending against power analysis by balancing binary values

a compiler based approach

Alexander Schlögl, supervised by Univ.-Prof. Dr. Rainer Böhme

Overview

Content

- Power analysis
- Approach
- Arithmetic
- Compiler Pass
- Results
- Future Work

Platform

Power analysis

Power analysis cont.

Power analysis cont.

Power analysis cont.

Approach

Approach cont.

Arithmetic

Arithmetic cont.

Find replacements for:

- `ORR`
- `AND`
- `XOR`
- `ADD`
- `SUB`
- `MUL`
- `SHIFTS`
- `DIV`
- `REM`

fig/placeholder.png

Verifying the arithmetic

Compiler pass

Compiler pass cont.

IR code before

fig/placeholder.png

IR code after

fig/placeholder.png

Binary operators

written as C functions

linked into same module

llvm operators changed to calls

Tradeoff

+ simplicity
+ modularity
+ small binaries
- (currently) on inlining
- overhead

rtlib.c

fig/placeholder.png

llvm-link

fig/placeholder.png

Balance.cpp

fig/placeholder.png

Evaluation

How to generate “virtual” power traces?

Qemu alone

- + fast
- - wrong resolution

Qemu + gdb

- + correct resolution
- + includes program location information
- - **very** slow

Execute instruction by instruction, dump registers every time

Results

	AES	
	unbalanced	balanced
No. of instructions	22 876	339 168
Relative increase	1	14.888
Balanced operations	20 571	334 521
Unbalanced operations	2211	4647
Balancedness	0.903	0.986
Code size	76 KB	78 KB

Results cont.

Results cont.

Results cont.

Results cont.

Results cont.

Results cont.

Future work

Same idea with different methods:

- Test on actual hardware
- Balance globals
- Mark balancing targets
- Move balancing to type system

Different ideas with same method:

- Other power analysis defenses
- Control flow randomization
- Move more security tools to LLVM

Conclusion