Master Thesis Proposal

Dual rail logic in software as LLVM-IR transformation

Alexander Schlögl

January 14, 2019

1 Introduction

Embedded devices very rarely utilize instruction level parallelism. Thus, as the power consumption is directly related to the bits in intermediate results that are set to 1, their power consumption directly reflects their computation results without much noise. If the device is running a cryptographic operation, this can result in a leakage of keys. This is known as a power analysis side channel attack[1].

While there exist many different defenses against this, both in software and in hardware, the most versatile of them is Dual-Rail-Logic[2]. Unlike most other defense mechanisms, Dual-Rail-Logic can be applied to any program, and works by calculating the inverse result $\bar{x}$ for each intermediate result x. This way, the power consumption (which is directly linked to the number of 1s in the result) is always the same, and the program is thus more robust against power analysis. Unfortunately, using Dual-Rail-Logic requires alterations to the hardware, and almost doubles the required circuitry size. This requirement makes it unsuitable for small embedded applications like e.g. SmartCards. In order to create a way of hardening *any* application against power analysis attacks, even when there are tight constraints on space, I would like to implement Dual-Rail-Logic in software.

To do this, I want to find a way to represent a balanced 8-bit arithmetic in a 32-bit architecture. While representing $\bar{x}$ and x should in theory only halve the word size, I will need additional space to represent carry bits and (new) intermediate steps in the registers as well, so the word size will probably be reduced to a quarter. The idea is to find a balancing scheme that allows me to perform all arithmetic and logic operations present in the intermediate representation (IR) of the LLVM compiler. Ideally, this scheme has no unbalanced intermediate results at all and utilizes no table lookups.

After finding such a balancing scheme and arithmetic, I want to transform the original code into balanced code in a custom LLVM optimization pass. This pass will transform the IR code of the original program into my balanced arithmetic operation by operation. Keeping the performance impact of this transformation as low as possible - both during compile and run-time - will be a major concern.

Finally I need a way of evaluating my work. For this I assume a perfect attacker capable of observing the power signature of every intermediate value. The robustness against such an attacker is then represented by the number of unbalanced values during the execution, as well as the ratio of balanced vs. unbalanced values.

To find this number I run the resulting code in the QEMU emulator, and observe the result of every operation. This allows me to easily test my work in a controlled environment and without any additional hardware.

The rest of this proposal is organized as follows: Section 2 covers Dual-Rail-Logic, as well as the LLVM and QEMU projects. In Section 3 I present my intended approach in full, and in Section 5 I discuss problems that might arise during implementation. Section 4 discusses previous work that has been done in similar directions.

2 Background

aoeu

3 Intended Methodology

aoeu

4 Related Work

aoeu

5 Possible Difficulties

aoeu

References

[1] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *Annual International Cryptology Conference*, pages 388–397. Springer, 1999.

[2] Danil Sokolov, Julian Murphy, Alexander Bystrov, and Alexandre Yakovlev. Design and analysis of dual-rail circuits for security applications. *IEEE Transactions on Computers*, 54(4):449–460, 2005.