Master Thesis Proposal

Dual rail logic in software as LLVM-IR transformation

Alexander Schlögl

December 6, 2018

Embedded devices do not run things in parallel. 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.

There exist many defenses against this form of attack, one of which is dual rail logic. In dual rail logic the bits are always balanced, i.e. the number of 1 bits is always constant. This is achieved by duplicating the circuitry and computing the result x as well as its inverse $\bar{x}$. While this is a very robust defense against power analysis, it also requires a substantial increase in circuitry size (in fact, the required size is almost doubled), which makes it unsuitable for small applications like e.g. key cards.

While there exist other, less space intensive defenses like masking, they are often algorithm specific and thus lack the generality of dual rail logic. A general way of creating machine instructions that are robust against power analysis for *any* code would be very desirable. To this end, I would like to develop a software implementation of dual rail logic. By representing 8bit logic in a 32bit architecture, I can hopefully achieve balanced intermediate results for all operations, making the resulting machine code more robust against power analysis attacks. I will achieve this by transforming operations in the intermediate representation (IR) generated by the LLVM compiler into balanced operations on a smaller word size.

My thesis will consist of three major parts, which will be discussed in the following:

1. Finding a suitable balanced arithmetic
2. Creating the transformation pass
3. Evaluating the result

Finding a suitable balanced arithmetic

TODO

Creating the transformation pass

The LLVM compiler has a human readable intermediate representation, and allows for easy addition of transformation passes over the IR of a program. It also has a variety of front-ends (source code to IR compilers), as well as back-ends (IR to machine code compilers), making the pass very general in terms of programming languages and architectures.

Evaluating the result

TODO