# Toward Balancing Arbitrary Code

June 11, 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[2].

While there exist many different defenses against this, both in software and in hardware, the most versatile of them is Dual-Rail-Logic[3]. 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 $1$s in the result) is always the same, and the program is thus more robust against power analysis. Unfortunately, using Dual-Rail-Logic requires a significant overhead, doubling the circuit size or more[1]. This requirement makes it unsuitable for small embedded applications like e.g. SmartCards.

To explore a new avenue in hardening against power analysis I have implemented a proof of concept (PoC) for Dual-Rail-Logic in software. This avoids the increase in required circuit size while still being applicable to arbitrary code.

The rest of this thesis is organized as follows: Section 2 gives an introduction to the tools used, as well as a brief refresher of the algorithms used for testing. Section 3 describes my approach, and Section 4 the implementation details of my thesis. Section 5 shows the evaluation results of my PoC. Finally, in Section 6 I offer a discussion of the results as well as an outlook to possible future work.

## 2 Background

### 2.1 Power Analysis Defenses

aoeu

### 2.2 LLVM

aoeu

### 2.3 QEMU

aoeu

### 2.4 AES

aoeu

### 2.5 RC4

aoeu

## 3 Methodology

aoeu

## 4 Implementation

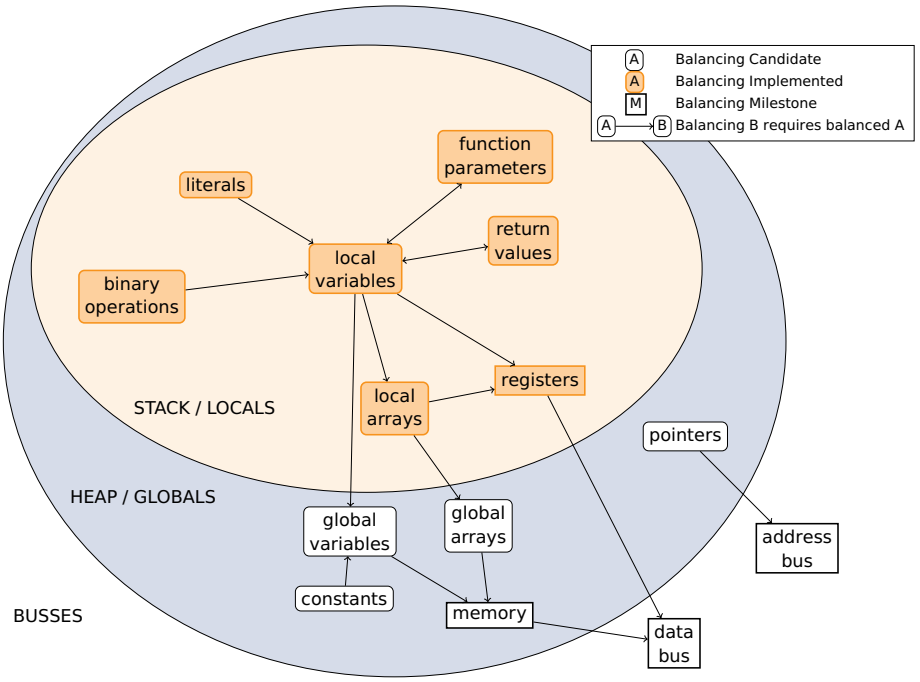
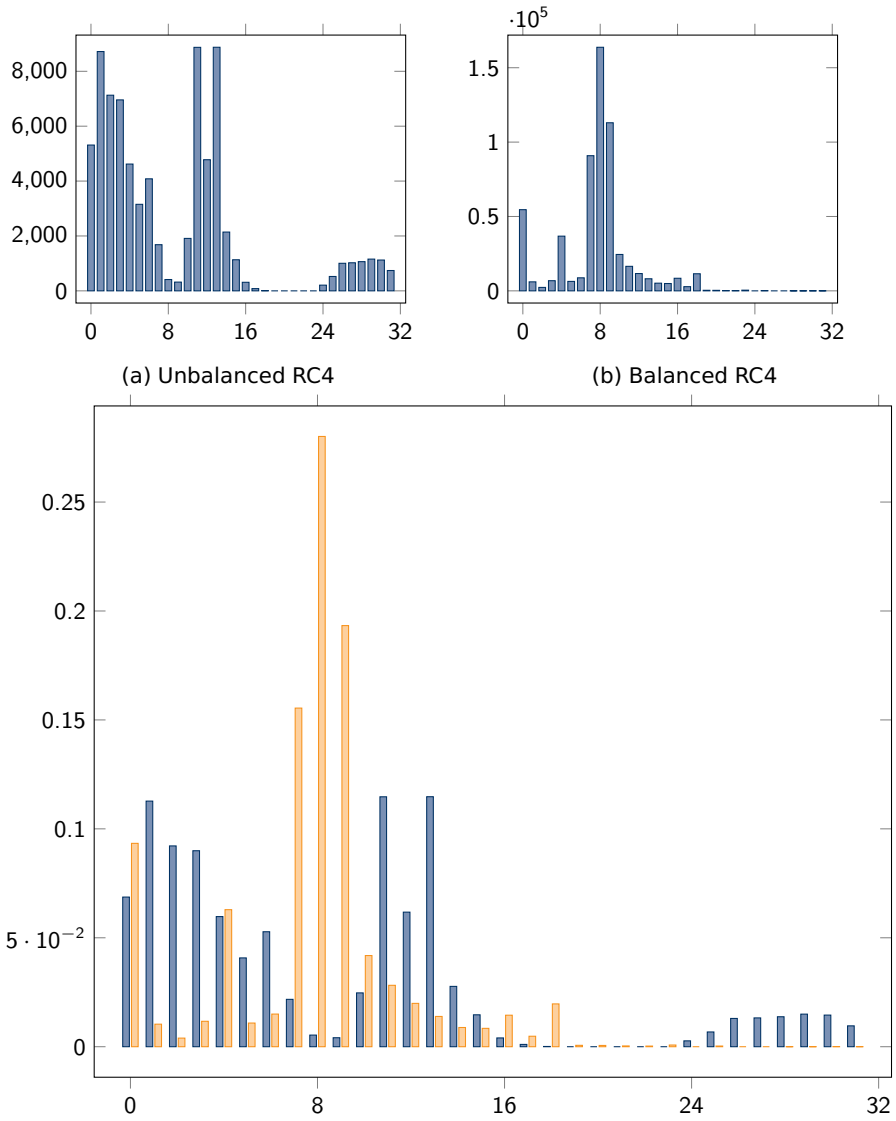



Figure 1: Balancing diagram for LLVM

## 5 Results

In this section I will discuss the balancing results for the two main algorithms I tested the pass on: RC4 and AES. Both algorithms have been written/adapted

so that they utilize the stack as much as possible, maximizing the benefit of my balancing pass.




(a) Unbalanced RC4

(b) Balanced RC4

(c) Scaled Hamming weight histograms for RC4

Figure 2: Hamming weight histograms for balanced and unbalanced RC4

# 6 Conclusion

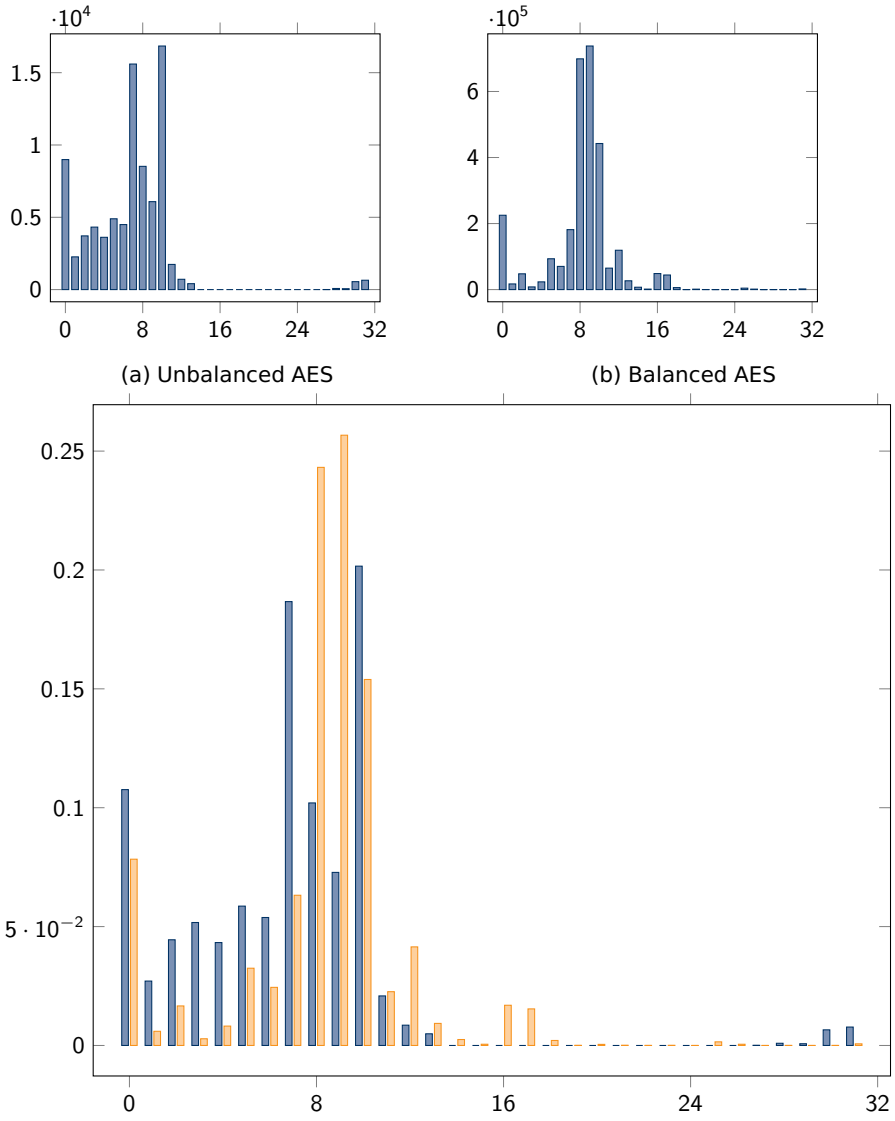
aoeu

## References


[1] Karthik Baddam and Mark Zwolinski. Path switching: a technique to tolerate dual rail routing imbalances. *Design Automation for Embedded Systems*, 12(3):207–220, 2008.

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

[3] 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.




(a) Unbalanced AES

(b) Balanced AES

(c) Scaled Hamming weight histograms for AES

Figure 3: Hamming weight histograms for balanced and unbalanced AES