Software Countermeasures Against DPA Attacks: Comparing Masking and Dual-Rail with Precharge Logic

Pablo Rauzy, Sylvain Guilley, and Jean-Luc Danger
TELECOM ParisTech
firstname.lastname@telecom-paristech.fr

There exist *side-channel attacks* (SCA for short) which target the software implementations of cryptosystems. This means that no parts of the software engineering process of a cryptosystem should be overlooked. Ideally, software should be bug-free and rely as little as possible on hand-written code for critical parts. These observations call for formally verified tools such as a certified compiler which would automatically add the necessary protections against SCA. Countermeasures can be classified in two categories [7]: on one hand those that use randomness to make the leakage statistically independent from sensitive data, for instance *masking* [5]; and on the other hand those that make the leakage indistinguishable, for instance *dual-rail with precharge logic* [6] (DPL for short). Automated masking has already been explored [9] but most efforts have yet to be done for DPL (despite the general-purpose CPU designed in DPL [4]).

Our aim is to list the technological barriers for automated code protection against side-channel attacks, to review the state-of-the-art and to investigate possible directions.

Even when implemented at the software level, masking and DPL countermeasures make assumptions about the hardware [11,1]. These hypotheses must be verified for the additional security to be operational. In masking, the shares should not interfere neither logically (for instance an opcode's effects should not depend on previously executed opcodes) nor physically (glitches, cross-coupling, self-demasking, *etc.*). In DPL, we must ascertain that (at least) two equivalent resources exists. These tests are often neglected, however, finding the minimal set of tests is not trivial. Formalizing them would allow to reach full coverage and *in fine* to automate them. We will detail the goal of these tests, and list a couple of them under some assumptions about the computation environment.

Conceptually, the above mentionned countermeasures are implemented on a modelization of the cryptographic algorithm. For masking, the modelization is a bit more difficult and restrictive because the data and operations in the algorithm must be embedded within a group (for instance $(\mathbb{F}_2^n, \oplus)$ for Boolean additive masking) which is specific to the masking scheme employed. The protection depends on the linearity of the operation; masking S-Boxes is difficult in general [10] despite some recent advances on generic masking schemes [8,3]. For DPL, the protections depends less on the algorithm. For example it can be bitsliced [2] which leads to a simple Turing Machine like model that operates at the bit level.

The most important aspect of this work is proofs. Our goals are twofold. First, we would like the compiler implementing the countermeasures to be formally proved (for instance using a model checker, or abstract interpretation), in the sense that it does not alter the semantics of the code it is protecting. The functionality of a cryptosystem should not be changed by securing it against SCA. Second, we'd like to be able to formally prove the efficiency of the added security. In this perspective, it is important to keep the code transformations and the model on which we work as simple as possible.

Efficiency in terms of speed at runtime is also an important issue. However an optimization of a protected piece of code should not damage the protection (i.e., losing the statistical independence in the case of masking, or the indistinguishability in the case of DPL). Most known code optimization techniques can be used (rescheduling, SAT, ...) but the optimizations must be security-aware. Formally proved code transformations and security can guarantee the validity of an optimization.

It has been shown that a masking protection induces a d^2 factor on the complexity of the protected algorithm, where d is the order of attack [3]. We are not aware of similar efficiency estimations for DPL. We have manually coded a DPL implementation of PRESENT in assembly language. This exercise gave us interesting feedback on the feasability of its automation. With respect to an unprotected implementation of PRESENT obtained by compilation, the execution time of our version is multiplied by approximately 20. We expect an automated tool to achieve much better results; the rationale for such a tool will be presented in the talk.

References

1. Shivam Bhasin, Sylvain Guilley, Youssef Souissi, and Jean-Luc Danger. Efficient FPGA Implementation of dual-rail countermeasures using Stochastic Models, September 26-27 2011. Non-Invasive Attack Testing Workshop (NIAT 2011), co-organized by NIST & AIST. Todai-ji Cultural Center, Nara, Japan. (`http://csrc.nist.gov/news_events/non-invasive-attack-testing-workshop/papers/06_Bhasin.pdf`PDF).
2. Eli Biham. A Fast New DES Implementation in Software. In *FSE*, pages 260–272, 1997.
3. Claude Carlet, Louis Goubin, Emmanuel Prouff, Michaël Quisquater, and Matthieu Rivain. Higher-Order Masking Schemes for S-Boxes. In *FSE*, pages 366–384, 2012.
4. Zhimin Chen, Ambuj Sinha, and Patrick Schaumont. Using Virtual Secure Circuit to Protect Embedded Software from Side-Channel Attacks. *IEEE Trans. Computers*, 62(1):124–136, 2013.
5. Jean-Sébastien Coron and Louis Goubin. On Boolean and Arithmetic Masking against Differential Power Analysis. In *CHES*, volume 1965 of *Lecture Notes in Computer Science*, pages 231–237. Springer, August 17-18 2000.
6. Philippe Hoogvorst, Guillaume Duc, and Jean-Luc Danger. Software Implementation of Dual-Rail Representation. In *COSADE*, 2011.
7. Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power Analysis Attacks: Revealing the Secrets of Smart Cards*. `http://www.springer.com/`Springer, December 2006. ISBN 0-387-30857-1, `http://www.dpabook.org/`.
8. Thomas S. Messerges. Using Second-Order Power Analysis to Attack DPA Resistant Software. In *CHES*, volume 1965 of *LNCS*, pages 238–251. Springer-Verlag, August 17-18 2000. Worcester, MA, USA.
9. Andrew Moss, Elisabeth Oswald, Dan Page, and Michael Tunstall. Compiler Assisted Masking. In *CHES*, pages 58–75, 2012.
10. Emmanuel Prouff and Matthieu Rivain. A Generic Method for Secure SBox Implementation. In *WISA*, pages 227–244, 2007.
11. Emmanuel Prouff and Thomas Roche. Higher-Order Glitches Free Implementation of the AES Using Secure Multi-party Computation Protocols. In *CHES*, pages 63–78, 2011.