

# Evaluation Report

Nicolai Müller

Ruhr University Bochum, Horst Görtz Institute for IT Security, Germany

E-Mail: nicolai.mueller@rub.de

Amir Moradi

University of Cologne, Institute for Computer Science, Germany

E-Mail: amir.moradi@uni-koeln.de

July 31, 2022

## 1 Target

1. Name: ascon\_v2
2. Algorithm: Ascon-128
3. Authors: Robert Primas and Rishub Nagpal (Institute of Applied Information Processing and Communications (IAIK), Graz University of Technology, Austria)
4. URL: [https://www.dropbox.com/s/8xygtv2u1hau9um/ASCON\\_IAIK.zip?dl=0&file\\_subpath=%2Fhardware%2Fascon\\_lwc%2Fsrc\\_rtl%2Fv2](https://www.dropbox.com/s/8xygtv2u1hau9um/ASCON_IAIK.zip?dl=0&file_subpath=%2Fhardware%2Fascon_lwc%2Fsrc_rtl%2Fv2)
5. Protection Method: Domain Oriented Masking [3]
6. Protection Order: 1

## 2 Setup

We evaluate the robust probing security of a given design, including the combined occurrence of glitches and transitions [1] by applying PROLEAD [4]. PROLEAD, a leakage detection tool publicly available at GitHub<sup>1</sup>, performs logic simulations at the gate level and applies statistical methods to evaluate the robust probing security of a circuit. For more information regarding PROLEAD, we refer to the PROLEAD wiki<sup>2</sup> and the original paper [4].

## 3 Evaluation

### 3.1 Netlist Generation

To generate a gate-level netlist from the given rtl source code, we synthesize the design with design compiler version O-2018.06-SP4 for linux64 - Nov 27, 2018. As the ASCON round function was already verified by the authors by using COCO [2], we choose `CryptoCore_SCA` as the top module to cover the whole crypto core. Hence, the evaluation covers the complete encryption of a single plaintext. As the given source code contains several parts whose syntax is not supported by the design compiler, we rewrote the concerned parts in `CryptoCore_SCA.vhd` and `design_pkg.vhd` in such a way that the syntax is supported by the design compiler. However, this does not affect the security or functionality of the design. The modified rtl code and the resulting gate-level netlist are given in the supplementary material.

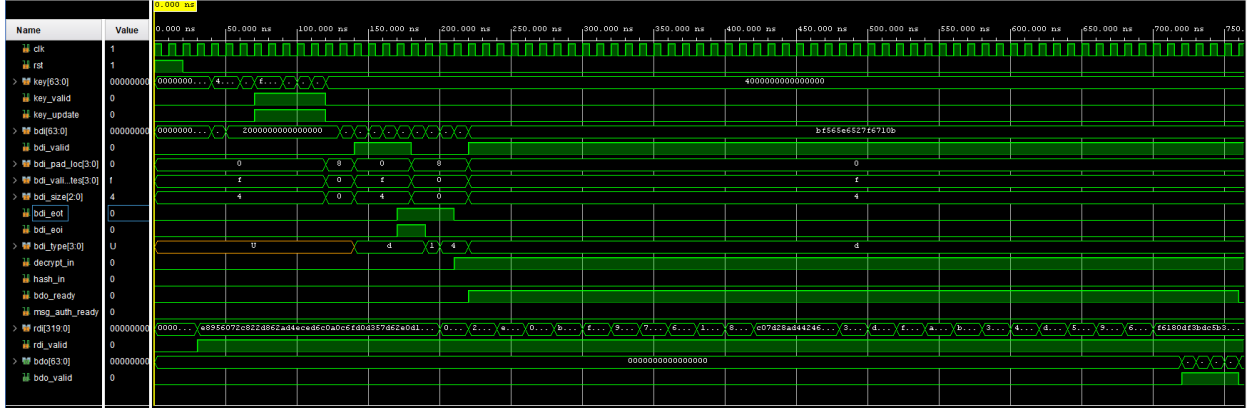
---

<sup>1</sup><https://github.com/ChairImpSec/PROLEAD>

<sup>2</sup><https://github.com/ChairImpSec/PROLEAD/wiki>

### 3.2 Evaluation Settings

To configure PROLEAD, we simulate the given design with Vivado 2020.2. Figure 1 shows the first 80 clock cycles of the simulation.



**Figure 1:** Simulation results encompassing the first key load and the first encryption procedure.

PROLEAD expects an initial sequence of inputs. Hence, we configure PROLEAD in a way that all inputs signals of `CryptoCore_SCA` are set the same as in the simulation. This means that PROLEAD initiates the run by loading a shared key and shared plaintext. Furthermore, we stop the simulation if the output signal `bdo_valid` goes high. This indicates that a single encryption procedure is finished. In short, PROLEAD simulates the loading and the complete encryption of a single plaintext. For the leakage detection, we perform a uniform and first-order fixed vs. random  $G$ -test conducting around 100 million simulations.

### 3.3 Results

For the first experiment, we evaluate the security under the glitch-extended probing model (without considering transitions). We detect strong first-order leakage by simulating around 1 million encryptions. The probe, causing the leakage, was placed on wire `n30226` which is the input wire of the register that stores the `msg_auth` signal. Hence, it becomes clear that a probe on `n30226` causes leakage as the tag checking procedure is unprotected. The results and the reports generated by PROLEAD are given in `results_compact0`. Since we want to ignore the unprotected tag checking for the following experiment, we exclude the probe on `n30226` from the evaluation. Moreover, we include the occurrence of transitions to the verification. We consider 100 million simulations and end up with a  $-\log_{10}(p)$ -value of 4.981022 which is under the internal 5.0 threshold. The results and the reports generated by PROLEAD are given in `results_compact1`. In summary, we did not find any leakage in the protected parts of the `CryptoCore_SCA` module. We, therefore, assume that the `CryptoCore_SCA` module (except for the tag checking procedure) is robust probing secure.

## References

- [1] FAUST, S., GROSSO, V., POZO, S. M. D., PAGLIALONGA, C., AND STANDAERT, F. Composable Masking Schemes in the Presence of Physical Defaults & the Robust Probing Model. *IACR TCHES 2018*, 3 (2018), 89–120.
- [2] GIGERL, B., HADZIC, V., PRIMAS, R., MANGARD, S., AND BLOEM, R. Coco: Co-Design and Co-Verification of Masked Software Implementations on CPUs. In *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021* (2021), USENIX Association, pp. 1469–1468.
- [3] GROSS, H., MANGARD, S., AND KORAK, T. Domain-Oriented Masking: Compact Masked Hardware Implementations with Arbitrary Protection Order. In *TIS@CCS 2016* (2016), B. Bilgin, S. Nikova, and V. Rijmen, Eds., ACM, p. 3.
- [4] MÜLLER, N., AND MORADI, A. PROLEAD - A Probing-Based Hardware Leakage Detection Tool. Cryptology ePrint Archive, Paper 2022/965, 2022. <https://eprint.iacr.org/2022/965>.