Mistakes Are Proof That You Are Trying: On Verifying Software Encoding Schemes' Resistance to Fault Injection Attacks

Jakub Breier, Dirmanto Jap, and Shivam Bhasin
Presenter: Wei He

Physical Analysis and Cryptographic Engineering
Nanyang Technological University, Singapore

PROOFS'16
20 August 2016

Table of Contents

Table of Contents

General Background

- The first software information hiding scheme was presented in 2011 by Hoogvorst et al.[1].
- They suggested to adopt the dual-rail precharge logic (DPL) to reduce the dependance of the power consumption on the data.
- Selmane et al.[2] showed that hardware DPL possesses properties that resist fault attacks naturally - however, this property has never been tested on software DPL.

[1] P. Hoogvorst, J.-L. Danger, and G. Duc. Software Implementation of Dual-Rail Representation, COSADE 2011.

[2] N. Selmane, S. Bhasin, S. Guilley, T. Graba, and J.-L. Danger. WDDL is Protected Against Setup Time Violation Attacks, FDTC'09.

Evaluated Proposals

In our work we examined three software encoding schemes:

- Bit sliced implementation of balanced assembly code that follows dual-rail precharge logic using look-up tables[3], *"Static-DPL XOR"* implementation.
- Balanced encoding achieved by adding complementary bits to processed data[4], *"Static-Encoding XOR"* implementation.
- Device-specific encoding, where the encoding function is selected based on device leakage[5], *"Device-Specific Encoding XOR"* implementation.

---

[3] P. Rauzy, S. Guilley, and Z. Najm. Formally Proved Security of Assembly Code Against Leakage, PROOFS 2014.

[4] C. Chen, T. Eisenbarth, A. Shahverdi, and X. Ye. Balanced Encoding to Mitigate Power Analysis: A Case Study, CARDIS 2014.

[5] H. Maghrebi, V. Servant, J. Bringer. There is wisdom in harnessing the strengths of your enemy: Customized encoding to thwart side-channel attacks, FSE 2016.

Contributions

- We analyzed three proposed software encoding countermeasures against faults attacks.
- We designed a code analyzer to understand the behavior of these countermeasures under common fault models.
- We performed a practical analysis, using laser fault injection equipment on an AVR, to validate the results from simulations.
- We highlighted weaknesses of the implementations and provided crucial insights on designing fault-resistant schemes.

"Static-DPL XOR" Implementation

- All the logical gates are implemented by using look-up tables (LUT) with balanced addressing.
- Bit-slicing – one byte carries only one bit of effective information.
- Only last two bits of each byte are used – 1 is encoded as 01 and 0 is encoded as 10.

Table: Look-up tables for *"DPL"* implementation.

index	0000 - 0100	0101	0110	0111 - 1000	1001	1010	1011 - 1111
and	00	01	10	00	10	01	00
or	00	01	01	00	01	10	00
xor	00	10	01	00	01	10	00

"Static-Encoding XOR" Implementation

- One byte carries 4 bits of information.
- Each nibble is balanced by adding complementary bits, in one of the two forms: $b_3\bar{b_3}b_2\bar{b_2}b_1\bar{b_1}b_0\bar{b_0}$ and $b_0\bar{b_2}b_1b_3\bar{b_1}b_2\bar{b_0}\bar{b_3}$.
- Following this rule, intermediate value at every point of time has Hamming weight 4.
- The scheme is explained on Prince cipher, which can be realized by using a balanced XOR and a balanced table-lookup – we have examined both operations.

"Device-Specific Encoding XOR" Implementation

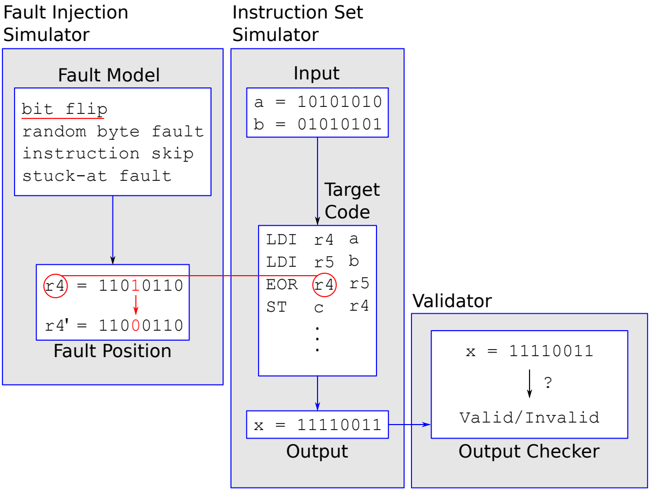
- Side-channel leakage is balanced by minimizing the variance of the encoded intermediate values.
- It is based on the fact that each register and each bit of register leaks the information differently.
- After the device profiling, the weight leakages β are used for calculating the encoding function.
- In this implementation, the length of codewords can be specified by the implementer.

Table of Contents

Fault Simulations

- Fault simulator was written in Java.
- We used 8-bit AVR microcontroller assembly code.
- The simulator injects faults in the target code under defined fault models.
- We considered inputs and outputs already encoded.
- We analyzed every instruction of code.

Fault Simulation Methodology



Inputs/Outputs

- *Static-DPL XOR:* uses inputs/outputs in format 00000001 for 1 and 00000010 for 0. There are only two possible values for a valid input, resulting in 4 different combinations of operands.
- *Static-Encoding XOR:* has inputs/outputs in format $a_3\bar{a_3}a_2\bar{a_2}a_1\bar{a_1}a_0\bar{a_0}$ and $b_3\bar{b_3}b_2\bar{b_2}b_1\bar{b_1}b_0\bar{b_0}$. Therefore, one variable in this encoding can take 16 different values, resulting in 256 input combinations.
- *Device-Specific Encoding XOR:* we used 8-bit implementation, using 16 codewords, resulting in 256 input combinations.

Outputs

We defined three possible output sets:

- *VALID* – output follows the proper encoding of each implementation.
- *INVALID* – output does not follow the proper encoding.
- *NULL* – output is all zero.

Fault Models

- *Single/multiple bitflip* – a content of the destination register of every operation was altered either to simulate single or multiple bit flip.
- *Instruction skip* – we skipped one or two instructions. Again, we tested all the possible combinations of instruction skips.
- *Random byte fault* – because of the specific encoding format, random byte faults are a subset of single/multiple bit flip faults.
- *Stuck-at fault* – we changed the content of the destination register of all the instructions in the code, one instruction at a time. We tested two values, all zeros and all ones.

Table of Contents

Experimental Setup

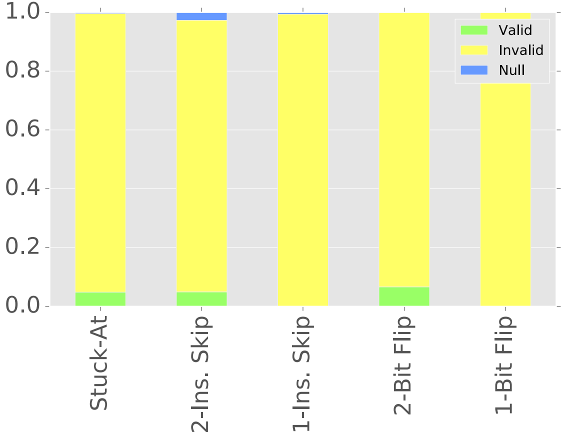
- DUT: Atmel ATmega328P microcontroller running at 16 MHz
- Laser: Infrared 1064 nm diode pulse laser.
- We found all the three kinds of faults, i.e. *INVALID*, *VALID* and *NULL*.
- The sensitive area of the chip is approximately $1100 \times 80\ \mu m^2$ large, out of $3 \times 3\ mm^2$ ($\approx 0.98\%$ of the whole chip area).

Results Overview

- We tested five different fault models and the faulty output could attain three possible states (*VALID, INVALID, NULL*).
- For *Static-Encoding XOR*, majority of the faults are *INVALID* for all fault models. Few faults are *VALID* and a negligible number of faults are *NULL* – this situation corresponds with experimental results.
- In *Static-Encoding LUT* and *Static-DPL XOR*, the simulations report a good mix of *INVALID* and *NULL* – however, number of *VALID* faults deviates from simulations to experiments.
- In *Device-Specific Encoding XOR*, there is slightly inflated number of *VALID* faults in experiments.

Static-Encoding XOR

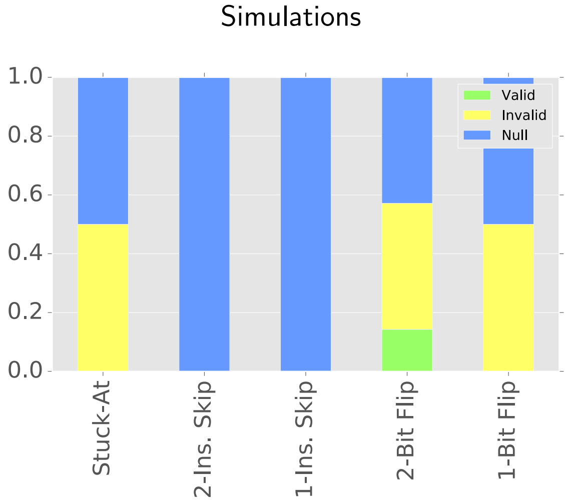
Simulations



Experiment

VALID	5.9%
INVALID	93.6%
NULL	0.6%

Static-Encoding LUT

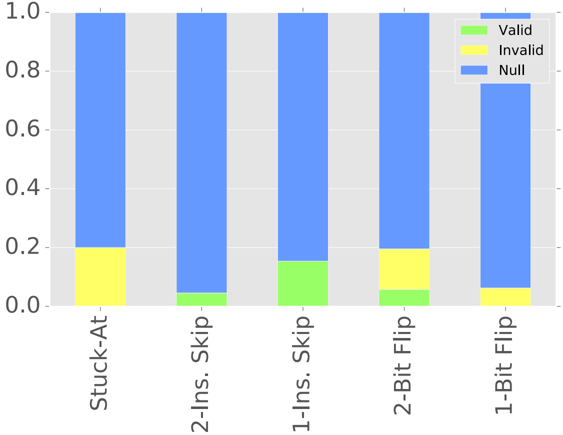


Experiment

VALID	32.4%
INVALID	47.7%
NULL	19.8%

Static-Encoding DPL

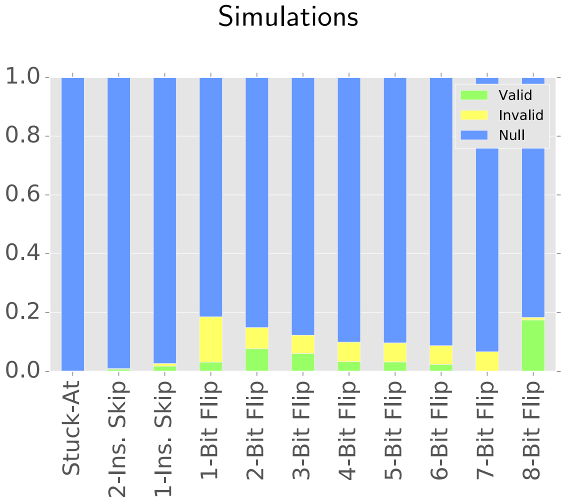
Simulations



Experiment

VALID	22.2%
INVALID	54.7%
NULL	23.1%

Device-Specific Encoding XOR



Experiment

VALID	13.57%
INVALID	3.86%
NULL	82.57%

Fault Propagation

- A *VALID* fault will always propagate to the output.
- Any *INVALID* or *NULL* input to *Static-DPL XOR*, *Static-Encoding LUT* and *Device-Specific XOR* will lead to a *NULL* at the output.
- *Static-Encoding XOR* does propagate faults – there are several combinations of inputs that lead to *VALID* output.
- Also, a combination of *NULL* input and *VALID* input leaks information about the input.

Table of Contents

Conclusion

- We have examined three software encoding proposals with respect to fault injection attacks – our results show weaknesses of these implementations.
- We simulated different fault models and validated our findings experimentally using laser fault injection station.
- In general, table look-up implementations offer higher level of security by thwarting the fault propagation.
- In comparison to hardware DPL, it takes significantly lower effort to disturb its software version using this fault model.
- *Device-Specific Encoding XOR* is currently the most secure scheme when it comes to fault attacks.

Thank you!

Any questions?

For further technical details, please contact:
jbreier@ntu.edu.sg