Fault Attacks Made Easy: Differential Fault Analysis Automation on Assembly Code

Jakub Breier, Xiaolu Hou
and Yang Liu

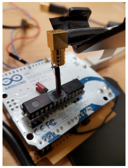
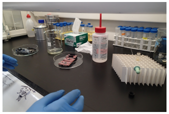
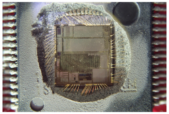
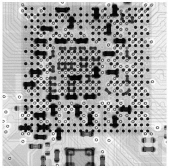
10 September 2018

Table of Contents

Table of Contents

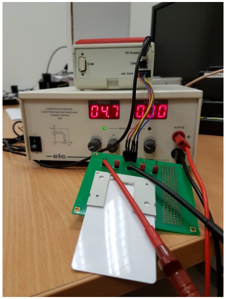
Fault Injection Attacks

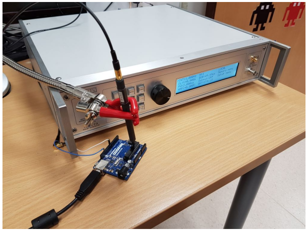
- Fault (injection) attacks are classified as semi-invasive physical attacks – often, a device depackaging is required.
- Exploit the possibility to insert a fault in the process of the algorithm execution in a way that could help to reveal the key.
- The idea of fault attacks was introduced by Boneh, DeMillo and Lipton in 1996[1].

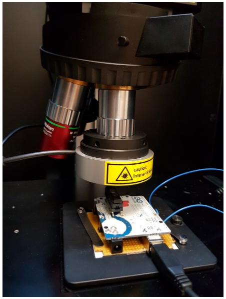





[1]D. Boneh, R. A. DeMillo, and R. J. Lipton. On the Importance of Checking Cryptographic Protocols for Faults, EUROCRYPT'97.

Fault Injection Techniques in Practice

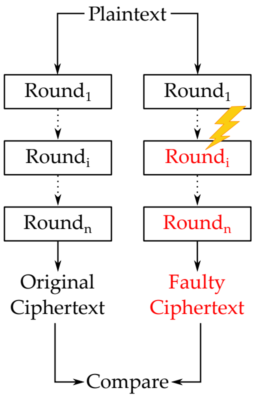

Voltage glitching


EM injection


Laser fault injection

Differential Fault Analysis

- One of the most popular FA techniques to attack symmetric block ciphers.
- Introduced by Biham and Shamir, targeting DES[2].
- Attacker injects a fault in a chosen round of the algorithm to get the desired fault propagation at the end of encryption.
- The secret key can then be determined by examining the differences between the correct and a faulty ciphertext.



[2] E. Biham and A. Shamir. Differential Fault Analysis of Secret Key Cryptosystems, CRYPTO'97.

Why Automation?

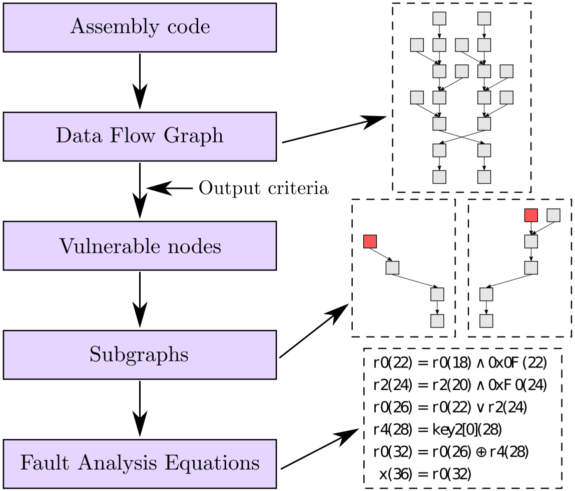
- All the current symmetric block ciphers have been shown vulnerable against fault attacks (especially DFA).
- The question is not whether the algorithm is secure or not, but which part of it is insecure.
- Automated methods can provide an answer fast and with minimal need of human intervention.

Why Automation on Assembly Code?

- In practice, the attack always has to be mounted on a real-world device.
- Different implementations of the same encryption algorithm do not necessarily share the same vulnerabilities.
- There might be an exploitable spot in the implementation that is not visible from the cipher design.
- There are works on fault analysis of a cipher from the cipher design level, there is no work aiming at DFA on the assembly code level.

Table of Contents

DATAC – DFA Automation Tool for Assembly Code



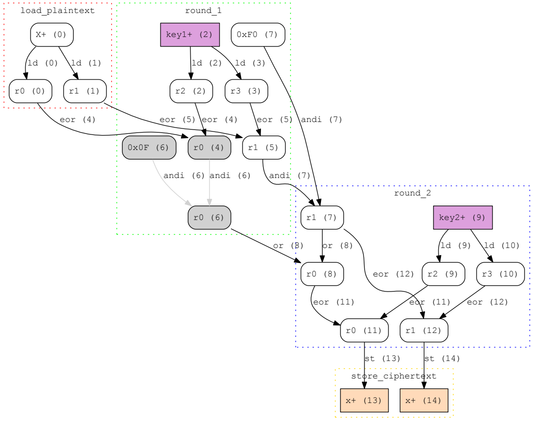
$$
\begin{aligned}
r0(22) &= r0(18) \wedge 0x0F(22)\\
r2(24) &= r2(20) \wedge 0xF0(24)\\
r0(26) &= r0(22) \vee r2(24)\\
r4(28) &= key2[0](28)\\
r0(32) &= r0(26) \oplus r4(28)\\
x(36) &= r0(32)
\end{aligned}
$$

Assumptions

- Known-ciphertext model and a single fault adversary.
- The implementation is available to the attacker and he can add annotations to the assembly code for the purpose of distinguishing different rounds, round keys, ciphertext words, etc
- For the analysis in this work, we have chosen Atmel AVR instruction set. However, for analyzing different instruction sets, only the parsing subsystem of the analyzer has to be redefined. Also, the methodology is universal.
- The implementation is unrolled, no direct/indirect jumps.

Example Program and Data Flow Graph

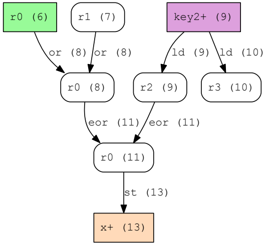
#	Instruction
	//load_plaintext
0	LD r0 X+
1	LD r1 X+
	//round_1
2	LD r2 key1+
3	LD r3 key1+
4	EOR r0 r2
5	EOR r1 r3
6	ANDI r0 0x0F
7	ANDI r1 0xF0
8	OR r0 r1
	//round_2
9	LD r2 key2+
10	LD r3 key2+
11	EOR r0 r2
12	EOR r1 r3
	//store_ciphertext
13	ST x+ r0
14	ST x+ r1



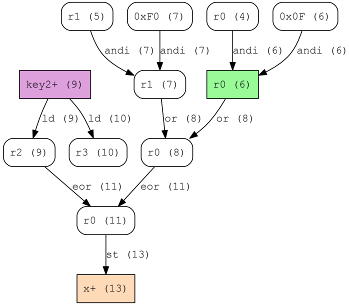
Output Criteria – Selection of Vulnerable Nodes

- `minAffectedCT`: minimal number of ciphertext nodes affected by the vulnerable node;
- `minDist`: minimal number of non-linear instructions between the node and a ciphertext node for at least `minAffectedCT` nodes;
- `maxDist`: maximum distance between the node and all the ciphertext nodes;
- `maxKey`: the number of the round keys, counting from the last round key, that are related to node a is at most `maxKey`;
- `minKeyWords`: there exists at least one round key such that the number of its corresponding key word nodes related to a is at least `minKeyWords`.

Subgraph Example



(a)

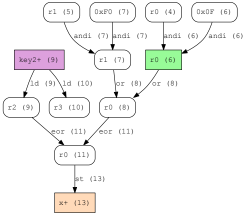


(b)

Subgraphs for node "r0 (6)" with depth (a) 0 and (b) 1, output criteria (`minAffectedCT`, `minDist`, `maxDist`, `maxKey`, `minKeyWords`)$= (1, 1, 1, 1, 1)$

DFA Equations Example

$$
\begin{aligned}
\text{“}\texttt{r0}\ (6)\text{”} &= \text{“}\texttt{r0}\ (4)\text{”} \wedge \text{“}\texttt{0x0F}\ (6)\text{”} && (1)\\
\text{“}\texttt{r1}\ (7)\text{”} &= \text{“}\texttt{r1}\ (5)\text{”} \wedge \text{“}\texttt{0xF0}\ (7)\text{”} && (2)\\
\text{“}\texttt{r0}\ (8)\text{”} &= \text{“}\texttt{r0}\ (6)\text{”} \vee \text{“}\texttt{r1}\ (7)\text{”} && (3)\\
\text{“}\texttt{r2}\ (9)\text{”} &= \texttt{key2[0]} && (4)\\
\text{“}\texttt{r0}\ (11)\text{”} &= \text{“}\texttt{r0}\ (8)\text{”} \oplus \text{“}\texttt{r2}\ (9)\text{”} && (5)\\
\text{“}\texttt{x+}\ (13)\text{”} &= \text{“}\texttt{r0}\ (11)\text{”}. && (6)
\end{aligned}
$$

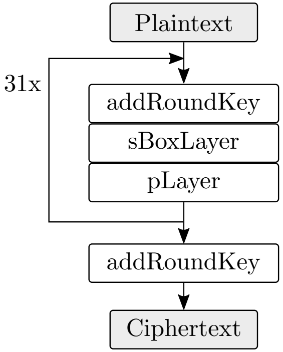


- (1): “$\texttt{r0}$ (6)” $= 0000 b_4 b_5 b_6 b_7$, $b_j \in \{0, 1\}$ $(j = 4, 5, 6, 7)$.
- (3): if we skip instruction 8, the result of (1) will be used in instruction 11 (5) instead of the result of (3)
- (4) and (6): the instruction skip attack on instruction 8 would result in the first four bits of `key2[0]` to appear as the first four bits of the faulted ciphertext.

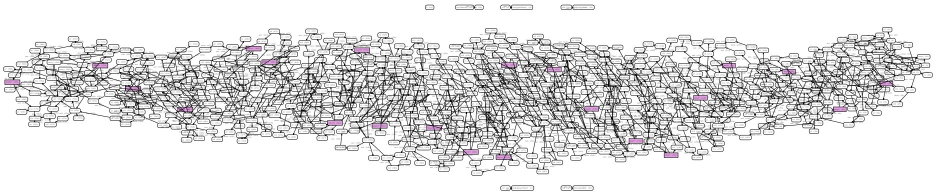
Table of Contents

PRESENT Cipher

- Block length: 64 bits
- Key length: 128 bits or 80 bits
- Based on SPN with following operations:
    - addRoundKey: `xor` with the round key
    - sBoxLayer: $4-$bit SBox
    - pLayer: bitwise permutation



Data Flow Graph of PRESENT – Full Version



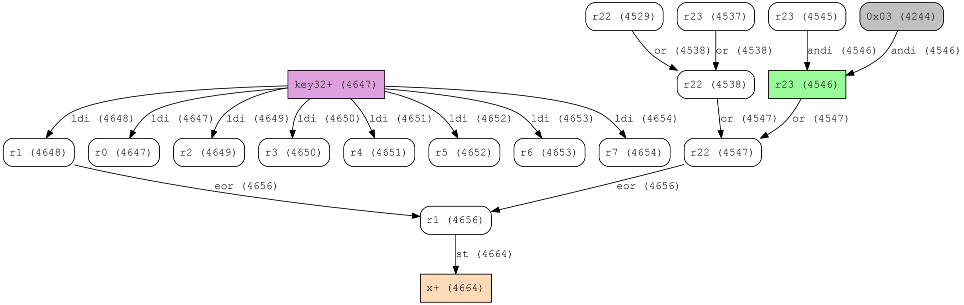
Unrolled implementation consists of over 4700 instructions.

New Attack found on PRESENT-80

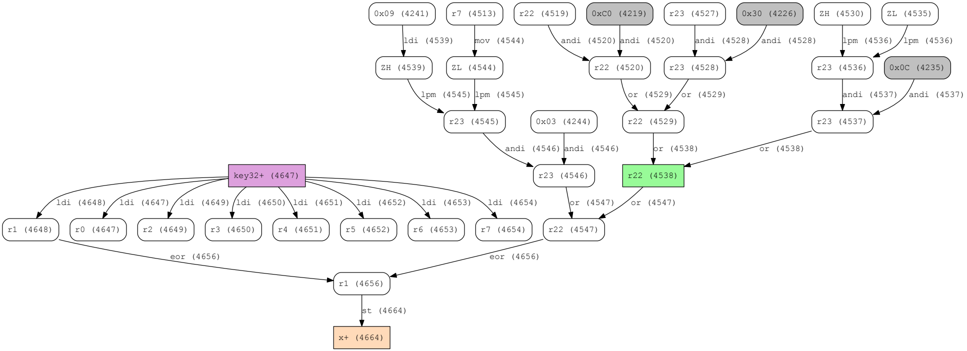
- We chose a speed-optimized assembly implementation for 8-bit AVR publicly available on GitHub[3].
- output criteria: (`minAffectedCT`, `minDist`, `maxDist`, `maxKey`, `minKeyWords`)$= (1,1,1,1,1)$.
- Recover the last round key by 16 fault injections.
- Implementation specific.
- Existing DFAs on PRESENT exploit Sbox look up which requires the analysis of the Sbox table by constructing DDT.
- Our new attack exploits `OR` operation which only requires the analysis of a simple truth table.

[3] `https://github.com/kostaspap88/PRESENT_speed_implementation`

New Attack found on PRESENT-80



New Attack found on PRESENT-80



Scalability of DATAC tested on AES with different number of rounds

# of rounds of AES	1	10	30	50
# of nodes	281	2,060	6,300	10,540
# of edges	415	3,209	9,909	16,609
# of instructions	259	1,901	5,801	9,701
Time (s)	0.07	0.87	5.11	38.89
Average time per round (s)	0.07	0.09	0.17	0.78
Memory (MB)	3	19	170	500

Data collected on laptop computer with mobile Intel Haswell family CORE i7 processor, 8 GB RAM

Table of Contents

Conclusion

- Proposed a methodology capable of finding spots vulnerable to DFA in software implementations of cryptographic algorithms.
- Created DATAC, which takes an assembly implementation and a user-specified output criteria as an input.
- DATAC outputs subgraphs for vulnerable nodes in the code, together with equations that can be directly used for DFA on the cipher implementation.
- New attacks on PRESENT-80, exploiting implementation-specific weaknesses.
- DATAC is scalable and can analyze current algorithms efficiently.

Thank you!

Any questions?

Web: `http://jbreier.com`

E-mail: `jakub.breier@gmail.com`