Enhanced Processor Defence Against Physical and Software Threats by Securing DIFT Against Fault Injection Attacks

PhD Defense

William PENSEC

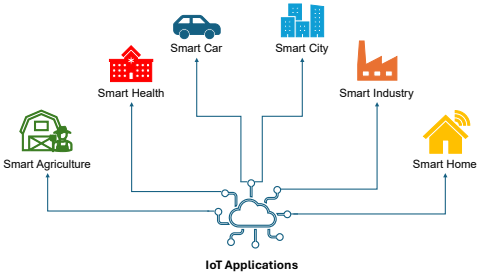
Université Bretagne Sud, UMR 6285, Lab-STICC, Lorient, France

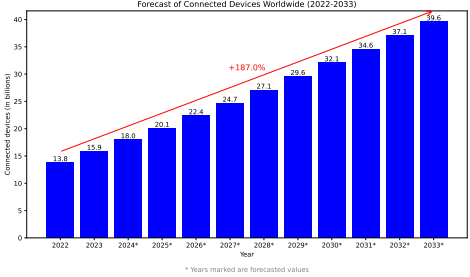
December 19, 2024

Context: Internet of Things

Internet of Things (IoT)

- Wide range of application
- Fast growing market with exponential usage
- Rely on sensors depending on their use
- Collect and share data
- Manipulation of critical data
- Increasingly vulnerable to multiple threats

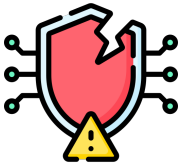




Motivations: IoT Under Threats

Threats

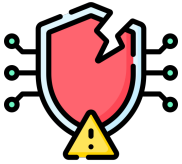
- Software threats: malwares, memory overflow attacks, SQL injection, etc
- Network threats: DDoS, Man-In-The-Middle, jamming, etc
- Hardware threats: physical attacks such as reverse engineering, Side-Channel Attacks (SCA), Fault Injection Attacks (FIA)



Motivations: IoT Under Threats

Threats

- **Software threats**: malwares, memory overflow attacks, SQL injection, etc
- Network threats: DDoS, man-in-the-middle, jamming, etc
- **Hardware threats**: physical attacks such as reverse engineering, Side-Channel Attacks (SCA), Fault Injection Attacks (FIA)



Software threats: Information Flow Tracking

- Security mechanism
- Protection against software attacks [1, 2]
- Static or Dynamic
- Software, Hardware or Hybrid
- Hardware DIFT: off-core, off-loading core, in-core

Software threats: Information Flow Tracking

- Security mechanism
- Protection against software attacks [1, 2]
- Static or Dynamic
- Software, Hardware or Hybrid
- Hardware DIFT: **off-core**, off-loading core, in-core

- Avantage: no internal hardware modification to the main core.

- Disadvantage: needs support from the OS for the synchronisation between data and tags.

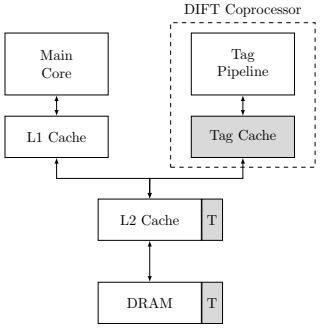


Figure 1: Representation of a Hardware Off-Core DIFT

Software threats: Information Flow Tracking

- Security mechanism
- Protection against software attacks [1, 2]
- Static or Dynamic
- Software, Hardware or Hybrid
- Hardware DIFT: off-core, **off-loading core**, in-core

- Avantage: hardware does not need to know DIFT tags and policies and no synchronisation is needed.

- Disadvantage: requires a multicore CPU, reducing the number of cores available and increase the power consumption.

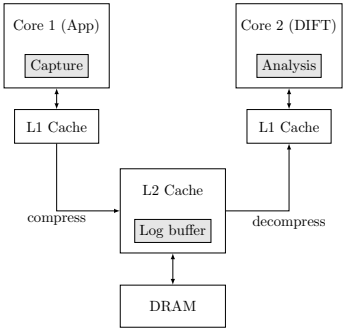


Figure 1: Representation of a Hardware Off-Loading DIFT

Software threats: Information Flow Tracking

- Security mechanism
- Protection against software attacks [1, 2]
- Static or Dynamic
- Software, Hardware or Hybrid
- Hardware DIFT: off-core, off-loading core, **in-core**

- Avantage: No multicore CPU and no synchronisation are needed. Very low performances overhead.

- Disadvantage: highly invasive modifications of internal hardware for tags computations and storing.

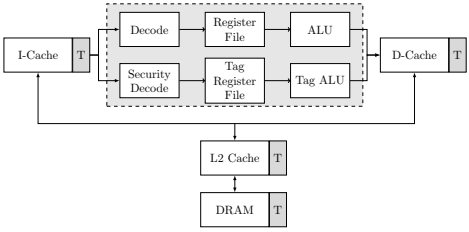
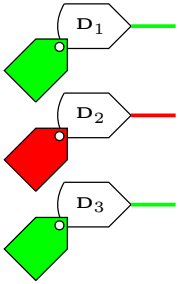


Figure 1: Representation of a Hardware In-Core DIFT

Dynamic Information Flow Tracking

Three steps

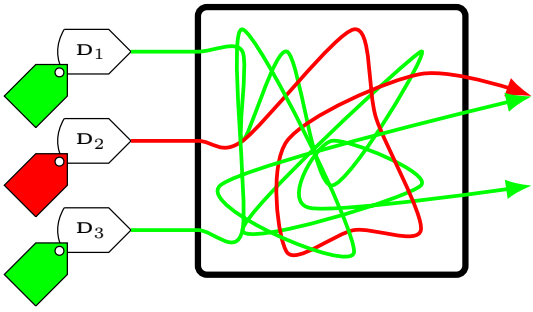
- Tag initialisation
- Tag propagation
- Tag check



Dynamic Information Flow Tracking

Three steps

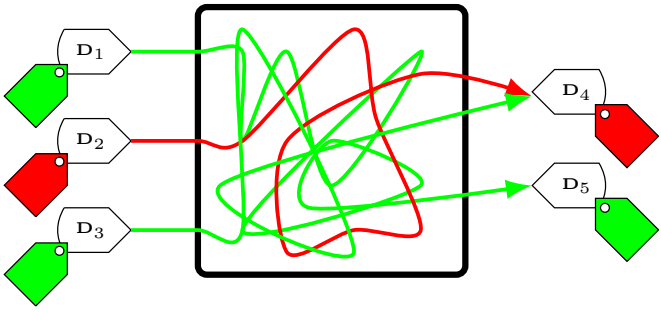
- Tag initialisation
- Tag propagation
- Tag check



Dynamic Information Flow Tracking

Three steps

- Tag initialisation
- Tag propagation
- Tag check

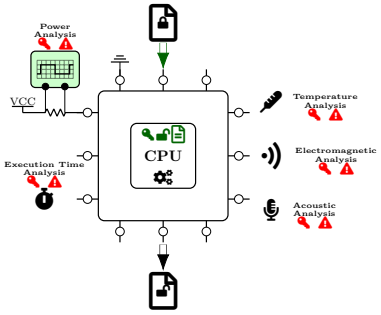


Hardware threats: Physical Attacks

- Reverse Engineering: process of information retrieval from a product by analysing and understanding the design, functionality, and operation of existing hardware
- Side-Channel Attacks: exploit information leakages on the circuit behaviour
- Fault Injection Attacks: involve deliberately introducing one or more fault(s) into the system to observe its behaviour and identify potential vulnerabilities.

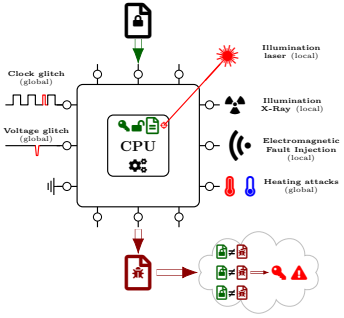
Hardware threats: Physical Attacks

- Reverse Engineering: process of information retrieval from a product by analysing and understanding the design, functionality, and operation of existing hardware
- Side-Channel Attacks: exploit information leakages on the circuit behaviour
- Fault Injection Attacks: involve deliberately introducing one or more fault(s) into the system to observe its behaviour and identify potential vulnerabilities.



Hardware threats: Physical Attacks

- Reverse Engineering: process of information retrieval from a product by analysing and understanding the design, functionality, and operation of existing hardware
- Side-Channel Attacks: exploit information leakages on the circuit behaviour
- Fault Injection Attacks: involve deliberately introducing one or more fault(s) into the system to observe its behaviour and identify potential vulnerabilities.



Issue

How can we maintain maximum protection against software attacks in the presence of physical attacks?

Objectives of this PhD Thesis

Contributions

- Provide a robust security mechanism against software and hardware threats.
- Take into account Fault Injection Attacks
- Propose lightweight countermeasures against FIA
- Take into account constraints, such as area and performance overhead

Outline

I. D-RI5CY – Vulnerability Assessment

II. Fault Injection Simulation for Security Assessment

III. Proposed protections against FIAs

IV. Experimental results

V. Conclusion and Perspectives

I. D-RI5CY – Vulnerability Assessment

D-RI5CY - origins

- Design [3] made by researchers at Columbia University (USA) with Politecnico di Torino (Italy)
- Based on the 32-bit RISC-V processor: RI5CY (Pulp Platform)
- Open source[1]
- 1-bit tag datapath
- Flexible security policy that can be modified at runtime

[1] https://github.com/sld-columbia/riscv-dift

D-RI5CY - architecture

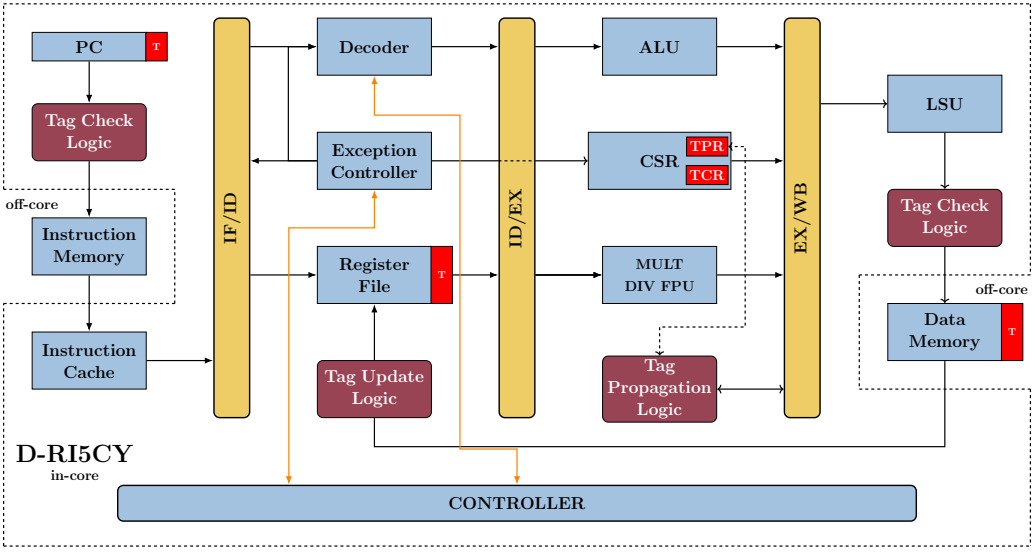


Figure 2: Architecture of the D-RI5CY.

Threat model

We consider an attacker able to:

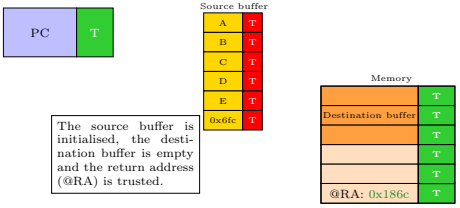
- perform a physical attack to defeat the DIFT mechanism and realise a software attack,
- inject faults in DIFT-related registers:
  - bit set,
  - bit reset,
  - bit-flip.

Methodology

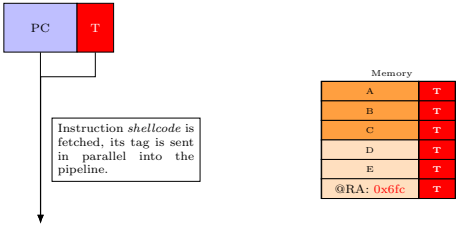
- Analysis of 3 use cases
- Published in Sensors S&P 2023 [4].

Case 1: Buffer overflow

- The attacker exploits a buffer overflow to access the return address register (*RA*).



(a) Malicious buffer and *RA* trusted



(b) Overflow and overwriting of *RA* and its tag

- As the data in the source buffer is manipulated by the user, it is marked as *untrusted*.
- Thanks to DIFT, the tags associated with the source buffer data overwrite the *RA* register tag.
- When the function returns, the corrupted register *RA* is loaded into *PC* using a jalr instruction.

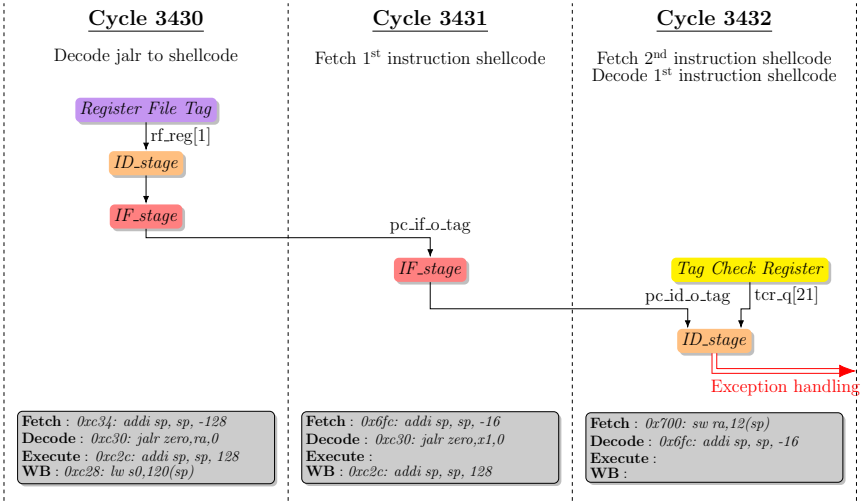


Figure 4: Temporal analysis of the tags propagation in *Buffer Overflow* attack

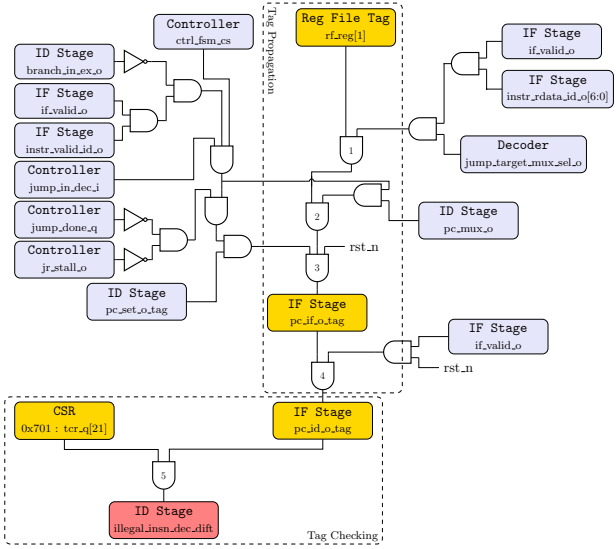


Figure 5: Logical analysis of the tags propagation in a *Buffer Overflow* attack

Experimental Setup - Simulation fault injections campaign

- Logical fault injection simulation is used for preliminary evaluations
  - faults are injected in the HDL code at cycle accurate and bit accurate level
  - a set of 55 DIFT-related registers are targeted
  - a reference simulation is done without fault
  - results are classed in four groups
    - crash: reference cycle count exceeded,
    - silent: current faulted simulation is the same as the reference simulation
    - delay: illegal instruction is delayed
    - success: DIFT has been bypassed
- Simulations with QuestaSim 10.6e.
- FISSA is used in order to create our injection campaigns

Buffer overflow

Table 1: End of simulation status

	Crash	NSTR	Delay	Success	Total
Buffer overflow	0	1380	20	24 (1.69%)	1422

Table 2: Buffer overflow : Register sensitivity as determined by fault model and simulation time

	Cycle 3428			Cycle 3429			Cycle 3430			Cycle 3431			Cycle 3432		
	set0	set1	bitflip	set0	set1	bitflip	set0	set1	bitflip	set0	set1	bitflip	set0	set1	bitflip
pc_if_o_tag										✓		✓			
memory_set_o_tag		✓	✓												
rf_reg[1]							✓		✓						
tcr_q	✓			✓			✓			✓			✓		
tcr_q[21]			✓			✓			✓			✓			✓
tpr_q	✓	✓		✓	✓										
tpr_q[12]			✓			✓									
tpr_q[15]			✓			✓									

Discussion

- 4266 simulations have been performed,
- 95 successes (2.23%),
- We have shown that the D-RI5CY DIFT is vulnerable to FIA
- Propagation of faults is facilitated by paths fully made of *AND* gates

II. Fault Injection Simulation for Security Assessment

Fault Injection Simulation for Security Assessment (FISSA)

Presentation

- Open-Source tool [2].
- Allows the circuit designer to analyse throughout the design cycle the sensibility against FIA.
- Integrated around an HDL Simulator (Questasim).
- The generated results can help to find vulnerabilities during the conceptual phase.
- FISSA enables the principles of Security by Design.
- Published in DSD 2024 [6].

[2] William Pensec. *FISSA: Fault Injection Simulation for Security Assessment*. URL: https://github.com/WilliamPsc/FISSA

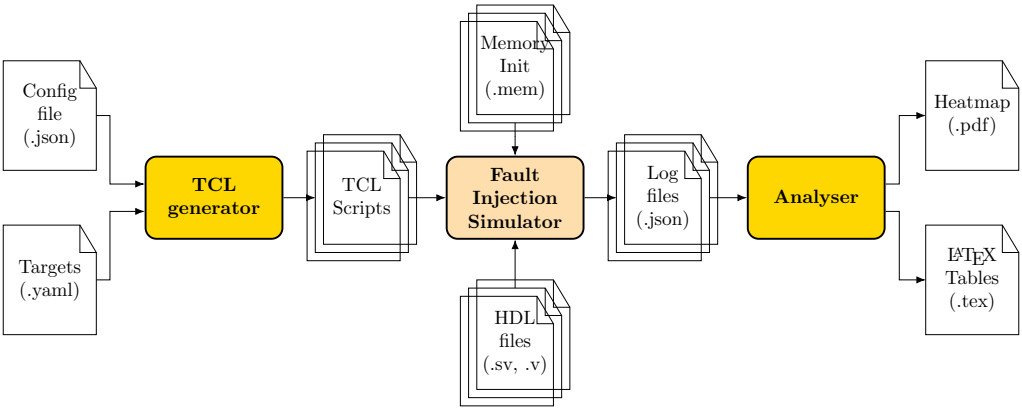


Figure 6: FISSA Architecture

FISSA — Supported Fault Models

Table 3: Supported Fault Model

Fault model	Number of target	Number of cycles	Target size
Set to 0	1	1	all
Set to 1	1	1	all
Single bit-flip in one target at a given clock cycle	1	1	all
Single bit-flip in two targets at a given clock cycle	2	1	all
Single bit-flip in two targets at two different clock cycles	2	2	all
Exhaustive multi-bits faults in one target at a given clock cycle	1	1	[1;16] bits
Exhaustive multi-bits faults in two targets at a given clock cycle	2	1	[1;10] bits

III. Proposed protections against FIAs

Protections

- Focusing into lightweight protections for small systems
- We propose 3 lightweight countermeasures using parity codes
  - Simple parity
  - Hamming Code
  - Hamming Code with an additional bit (SECDED)

Detection of single-bit errors - Simple Parity

- Often used for error detection.
- Add an extra bit for parity computation.
- Can only detect one error without correction.

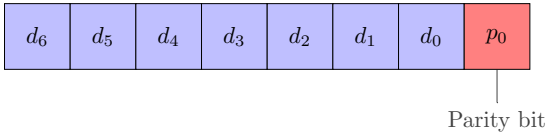


Figure 7: Simple parity codeword

Implementation - Simple Parity

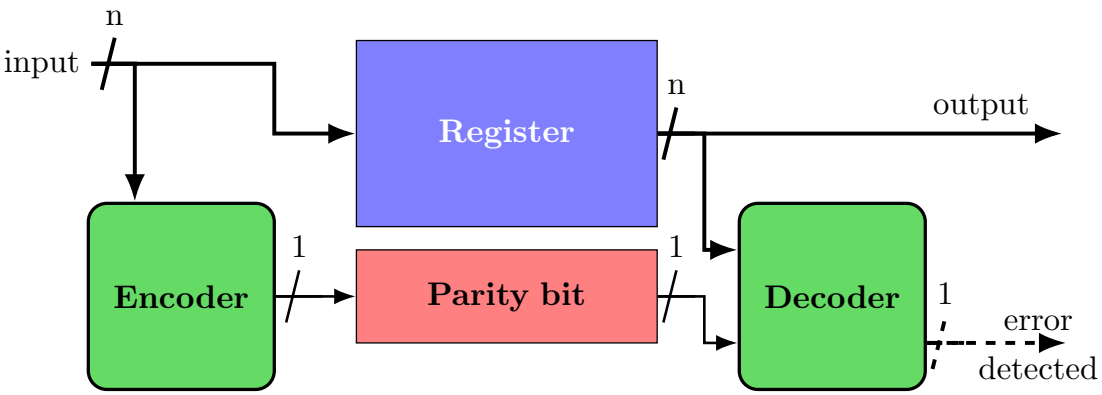


Figure 8: Simple Parity implementation

Detection and correction of single-bit errors - Hamming Code

- Linear error-correcting codes invented by Richard W. Hamming [7].
- Mostly used in digital communication and data storage systems.
- Detect and correct single-bit error.
- Redundancy bits are placed in power of 2 positions.
- The number of redundancy bits depends on the number of bits to be protected $(2^r \geq m + r + 1)$

$$
\begin{aligned}
r_0 &= d_0 \oplus d_1 \oplus d_3 \oplus d_4 \oplus d_6 \\
r_1 &= d_0 \oplus d_2 \oplus d_3 \oplus d_5 \oplus d_6 \\
r_2 &= d_1 \oplus d_2 \oplus d_3 \\
r_3 &= d_4 \oplus d_5 \oplus d_6
\end{aligned}
\tag{1}
$$

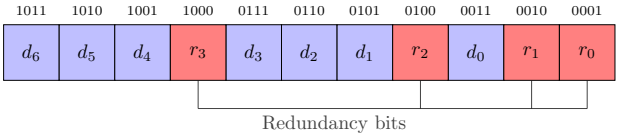


Figure 9: Hamming codeword

Detection of two-bit errors and correction of single-bit errors - SECDED

- Based on Hamming Code.
- Detect two-bit error and correct single-bit error.
- An additional bit is added: general parity bit

$$r_0 = d_0 \oplus d_1 \oplus d_3 \oplus d_4 \oplus d_6$$
$$r_1 = d_0 \oplus d_2 \oplus d_3 \oplus d_5 \oplus d_6$$
$$r_2 = d_1 \oplus d_2 \oplus d_3$$
$$r_3 = d_4 \oplus d_5 \oplus d_6$$
$$gp_0 = \bigoplus_{i=0}^{6} d_i \oplus \bigoplus_{j=0}^{3} r_j \quad (2)$$

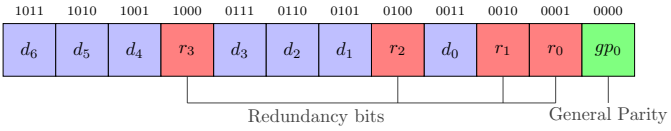


Figure 10: SECDED codeword

Implementation - SECDED

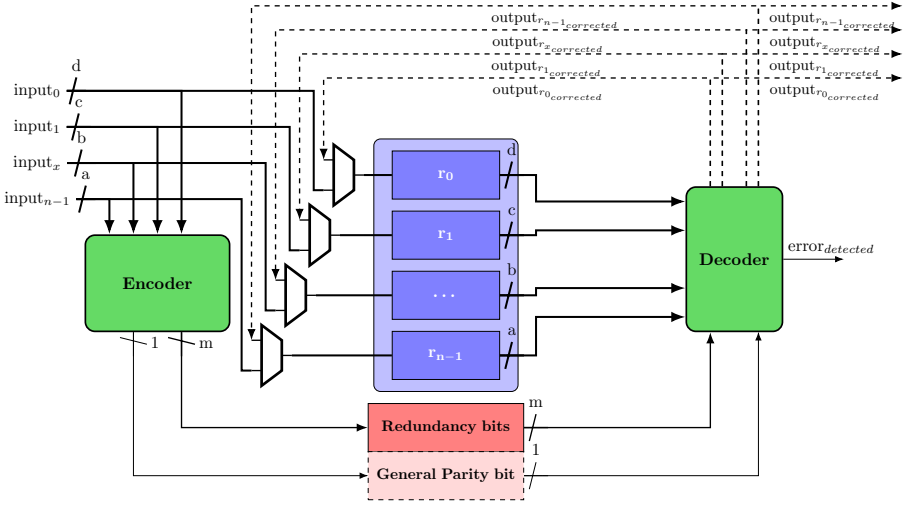


Figure 11: SECDED implementation for independant registers

Implementation - SECDED

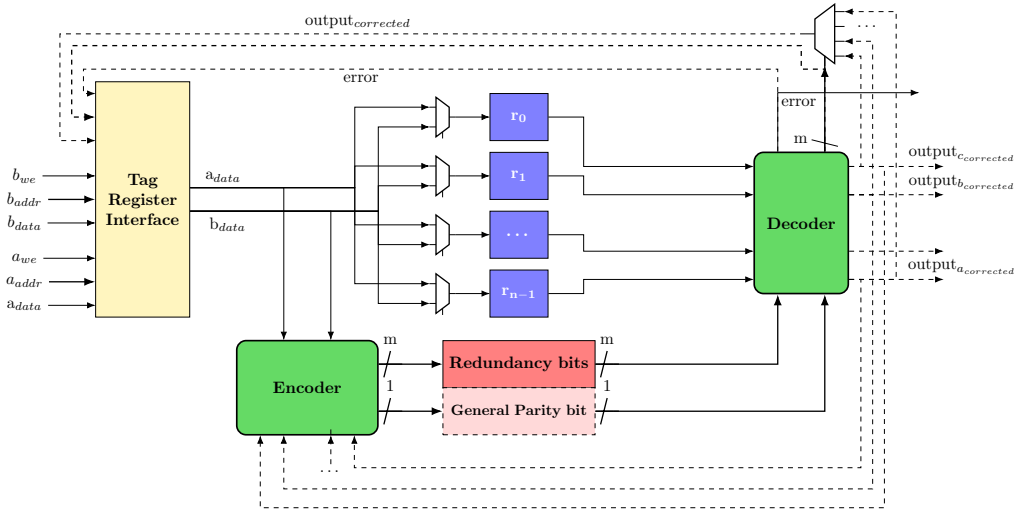


Figure 12: SECDED implementation for Register File

Threat model

- DIFT-related registers + protection-related registers
- Single bit-flip in one register
- Single bit-flip in two registers at two distinct clock cycles
- Single bit-flip in two registers at a given clock cycle
- Multi-bit faults in one register at a given clock cycle
- Multi-bit faults in two registers at a given clock cycle

Implemented strategies - Group composition

Table 4: Grouping composition of implemented strategies

	Grouping strategy	Number of groups	Number of registers
Baseline – D-RI5CY	–	–	55
Strategy 1	Minimisation of groups	5	65
Strategy 2	Protection per pipeline stage	7	69
Strategy 3	Protection per register	24	103
Strategy 4	Protection per register with slicing of CSR	38	131
Strategy 5	Coupling sliced registers	39	133

Implemented strategies - details

Table 5: Summary of DIFT-related protected registers – taking SECDED

	Number of protected bits	Number of redundancy bits	Number of parity bits	Number of bits
Strategy 1	107	25	5	157
Strategy 2	107	30	7	164
Strategy 3	107	64	24	215
Strategy 4	103	101	38	266
Strategy 5	102	114	39	280

IV. Experimental results

Experimental setup

- Use of FISSA for FIA campaigns
- Few results presented here

FPGA implementation results

Table 6: FPGA implementation results — Vivado 2023.2

Protection	Number of LUTs	Number of FFs	Maximum frequency
D-RI5CY	6911 (0%)	2335 (0%)	47.6 MHz (0%)
Hamming Code Strategy 1	7283 (5.38%)	2361 (1.11%)	47.4 MHz (-0.36%)
Hamming Code Strategy 2	7369 (6.63%)	2363 (1.2%)	46.9 MHz (-1.43%)
Hamming Code Strategy 3	7251 (4.92%)	2361 (1.11%)	46.8 MHz (-1.67%)
Hamming Code Strategy 4	7203 (4.23%)	2371 (1.54%)	47.6 MHz (0%)
Hamming Code Strategy 5	7182 (3.92%)	2411 (3.25%)	47.3 MHz (-0.57%)
SECDED Strategy 1	7428 (7.48%)	2366 (1.33%)	47.2 MHz (-0.95%)
SECDED Strategy 2	7433 (7.55%)	2366 (1.41%)	47.2 MHz (-0.95%)
SECDED Strategy 3	7324 (5.98%)	2368 (1.28%)	47.5 MHz (-0.24%)
SECDED Strategy 4	7255 (4.98%)	2365 (1.93%)	48.3 MHz (1.43%)
SECDED Strategy 5	7228 (4.59%)	2428 (3.98%)	48.3 MHz (1.43%)

Table 7: Logical fault injection simulation campaigns results for exhaustive multi-bits faults in two registers at a given clock cycle

		Crash	Silent	Delay	Detection	Detection & Correction	Double Error Detection	Success	Total	Execution time (h:min)
Buffer Overflow	No protection	0	67 072	926	–	–	–	450 (0.66%)	68 448	11:11
	Simple parity	0	24 622	8	53 359	–	–	59 (0.08%)	78 048	25:00
	Hamming 1	0	294 464	6273	–	–	–	3103 (1.02%)	303 840	99:36
	Hamming 2	0	0	3992	–	319 588	–	4356 (1.33%)	327 936	131:12
	Hamming 3	0	0	4557	–	436 187	–	4408 (0.99%)	445 152	121:20
	Hamming 4	0	0	5446	–	590 953	–	5329 (0.89%)	601 728	167:00
	Hamming 5	0	0	5987	–	714 873	–	5860 (0.81%)	726 720	210:31
	SECDED 1	0	0	1911	–	150 791	170 575	723 (0.22%)	324 000	86:59
	SECDED 2	0	0	1186	–	170 805	184 761	584 (0.16%)	357 336	94:04
	SECDED 3	0	0	1230	–	300 260	263 665	669 (0.12%)	565 824	161:30
	SECDED 4	0	0	18	–	457 498	368 959	61 (0.01%)	826 536	244:48
	SECDED 5	0	0	39	–	576 992	401 407	66 (0.01%)	978 504	284:45

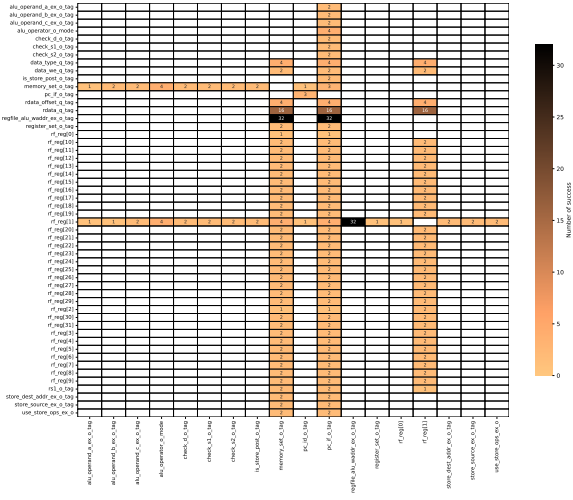


Figure 13: Unprotected version

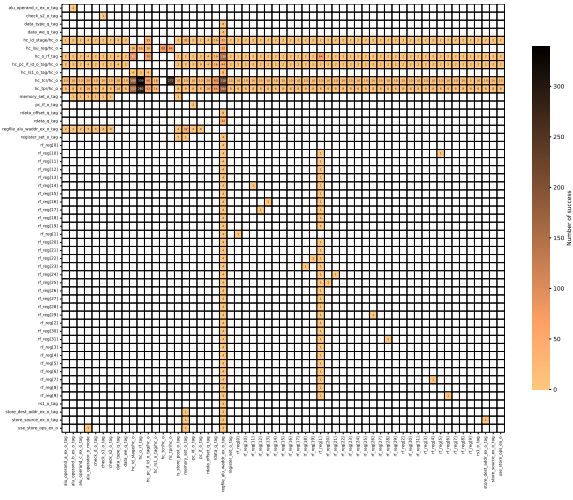


Figure 14: Hamming Code 2 protected version

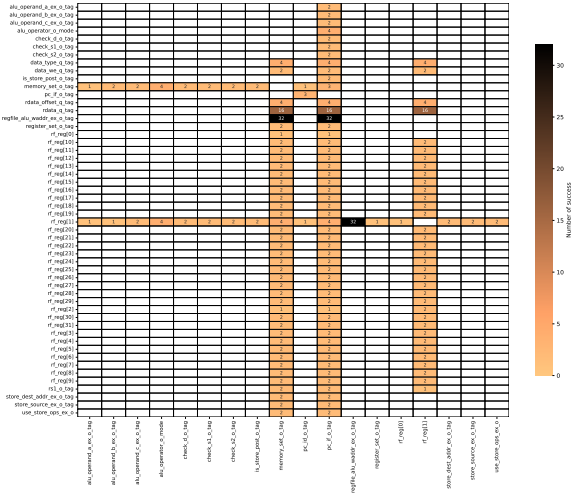


Figure 13: Unprotected version

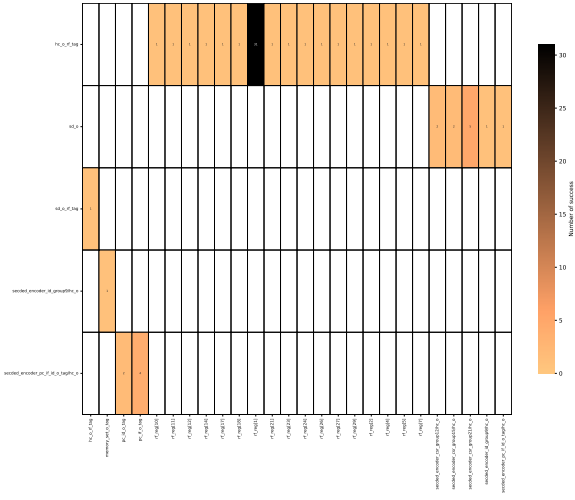


Figure 14: SECDED 5 protected version

V. Conclusion and Perspectives

Conclusion

Presented:

- Vulnerability assessment of a DIFT mechanism against FIA.
- Proposition of 3 lightweight countermeasures to protect against different fault models.
- Open-Source tool to help find vulnerabilities during the conceptual phase.

Perspectives

- Extend the assessment of more complex DIFT
- Further development of FISSA
- Propose more robust countermeasures to correct multiple faults
- Conduct real-world FIA

Contributions

International peer-reviewed conferences with proceedings

1. **William Pensec**, Vianney Lapôtre, and Guy Gogniat. 2023. Another Break in the Wall: Harnessing Fault Injection Attacks to Penetrate Software Fortresses. In Proceedings of the First International Workshop on Security and Privacy of Sensing Systems (SensorsS&P), 2023. [4]
2. **William Pensec**, Francesco Regazzoni, Vianney Lapôtre, and Guy Gogniat. Defending the Citadel: Fault Injection Attacks Against Dynamic Information Flow Tracking and Related Countermeasures. 2024 IEEE Computer Society Annual Symposium on VLSI (ISVLSI), 2024, pp. 180-185. [8]
3. **William Pensec**, Vianney Lapôtre, and Guy Gogniat. Scripting the Unpredictable: Automate Fault Injection in RTL Simulation for Vulnerability Assessment. 2024 27th Euromicro Conference on Digital System Design (DSD), 2024. [6]

Contributions

Conferences without proceedings

1. Vianney Lapôtre, **William Pensec** and Guy Gogniat, When in-core Dynamic Information Flow Tracking faces fault injection attacks, 19th International Workshops on Cryptographic architectures embedded in logic devices (CryptArchi), Cantabria, Spain, June 2023, https://hal.science/hal-04381235
2. **William Pensec**, Vianney Lapôtre and Guy Gogniat, Unveiling the Invisible Threads: Dynamic Information Flow Tracking and the Intriguing World of Fault Injection Attacks, Journée thématique sur les Attaques par Injection de Fautes (JAIF), Gardanne, September 2023, https://hal.science/hal-04727439

Popularising science event

- Participation in a science outreach event, "*Ma thèse en 180 secondes*" ("My Thesis in 180 seconds"), Rennes, March 2023, https://youtu.be/m_whL8xGbMQ

Enhanced Processor Defence Against Physical and Software Threats by Securing DIFT Against Fault Injection Attacks

PhD Defense

William PENSEC

Thank you for your attention.

Composition of the Jury

Examiners:	Jean-Max DUTERTRE
	Francesco REGAZZONI
Reviewers:	Lejla BATTINA
	Nele MENTENS
	Vincent BEROULLE
PhD supervisor:	Guy GOGNIAT
Thesis co-director:	Vianney LAPÔTRE

References

References

[1] Christopher Brant et al. “Challenges and Opportunities for Practical and Effective Dynamic Information Flow Tracking”. In: *ACM Computing Surveys* 55.1 (Nov. 2021). ISSN: 0360-0300. DOI: 10.1145/3483790.

[2] Wei Hu, Armaiti Ardeshiricham, and Ryan Kastner. “Hardware Information Flow Tracking”. In: *ACM Computing Surveys* (2021). DOI: 10.1145/3447867.

[3] Christian Palmiero et al. “Design and Implementation of a Dynamic Information Flow Tracking Architecture to Secure a RISC-V Core for IoT Applications”. In: *High Performance Extreme Computing*. 2018. DOI: 10.1109/HPEC.2018.8547578.

[4] William Pensec, Vianney Lapôtre, and Guy Gogniat. “Another Break in the Wall: Harnessing Fault Injection Attacks to Penetrate Software Fortresses”. In: *Proceedings of the First International Workshop on Security and Privacy of Sensing Systems*. SensorsS&P. Istanbul, Turkiye: Association for Computing Machinery, 2023, pp. 8–14. DOI: 10.1145/3628356.3630116.

[5] William Pensec. *FISSA: Fault Injection Simulation for Security Assessment*. URL: https://github.com/WilliamPsc/FISSA.

[6] William Pensec, Vianney Lapôtre, and Guy Gogniat. “Scripting the Unpredictable: Automate Fault Injection in RTL Simulation for Vulnerability Assessment”. In: *2024 27th Euromicro Conference on Digital System Design (DSD)*. Paris, France, Aug. 2024, pp. 369–376. DOI: 10.1109/DSD64264.2024.00056.

[7] R. W. Hamming. “Error detecting and error correcting codes”. In: *The Bell System Technical Journal* (1950). DOI: 10.1002/j.1538-7305.1950.tb00463.x.

References

[8] William PENSEC et al. "Defending the Citadel: Fault Injection Attacks Against Dynamic Information Flow Tracking and Related Countermeasures". In: *2024 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. Knoxville, United States, July 2024, pp. 180–185. DOI: 10.1109/ISVLSI61997.2024.00042.

[9] Transforma Insights; Exploding Topics. *Number of Internet of Things (IoT) connections worldwide from 2022 to 2023, with forecasts from 2024 to 2033*. Online. Accessed 13 August 2024. 2024. URL: https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/.

Backup

Tag Propagation Register

Table 8: Tag Propagation Register configuration

	Load/Store Enable	Load/Store Mode	Logical Mode	Comparison Mode	Shift Mode	Jump Mode	Branch Mode	Arith Mode
Bit index	17 16 15	13 12	11 10	9 8	7 6	5 4	3 2	1 0
Policy V1	0 0 1	1 0	1 0	0 0	1 0	1 0	0 0	1 0
Policy V2	1 1 1	1 0	1 0	1 0	1 0	1 0	1 0	1 0

- A Mode field for each class of instructions which specifies how to propagate the tags of the input operands to the output operand tag.
  - the output tag keeps its old value (00);
  - the output tag is set to one, if both the input tags are set to one (01);
  - the output tag is set to one, if at least one input tag is set to one (10);
  - the output tag is set to zero (11).
- The three bits in the L/S enable field allow the policy to enable the source, source-address, and destination-address tags, respectively

Tag Check Register

Table 9: Tag Check Register configuration

	Execute Check	Load/Store Check	Logical Check	Comparison Check	Shift Check	Jump Check	Branch Check	Arith Check
Bit index	21	20 19 18 17	16 15 14	13 12 11	10 9 8	7 6 5	4 3	2 1 0
Policy V1	1	1 0 1 0	0 0 0	0 0 0	0 0 0	0 0 0	0 0	0 0 0
Policy V2	0	0 0 0 0	0 0 0	0 0 0	0 0 0	0 0 0	0 0	0 1 1

- The tag-check rules restrict the operations that may be performed on tagged data. If the check bit for an operand tag is set to one and the corresponding tag is equal to one, an exception is raised.
  - For all the classes except Load/Store, there are three tags to consider: first input, second input, and output tags
  - For the Load/Store class there are four tags to take into account: source-address, source, destination-address, and destination tags
  - the additional Execute Check field is associated with the program counter and specifies whether to raise a security exception when the program-counter tag is set to one

Case 2: WU-FTPd

- The vulnerability is the use of an unchecked user input as the format string parameter in functions that perform formatting, e.g. printf()
- An attacker can use the format tokens, to write into arbitrary locations of memory, e.g. the return address of the function.

```
void echo(){
    int a;
    register int i asm("x8");
    a = i;
    printf("%224u%n%35u%n%253u%n%n", 1, (int*) (a-4), 1, (int*) (a-3), 1, (int*) (a-2), (int*) (a-1));
}
```

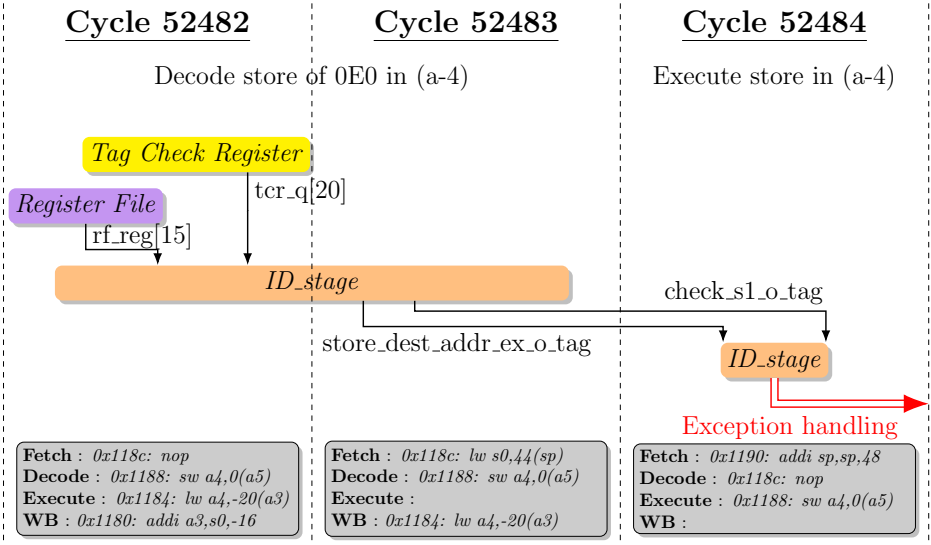


Figure 15: Temporal analysis of the tags propagation in a *format string* attack

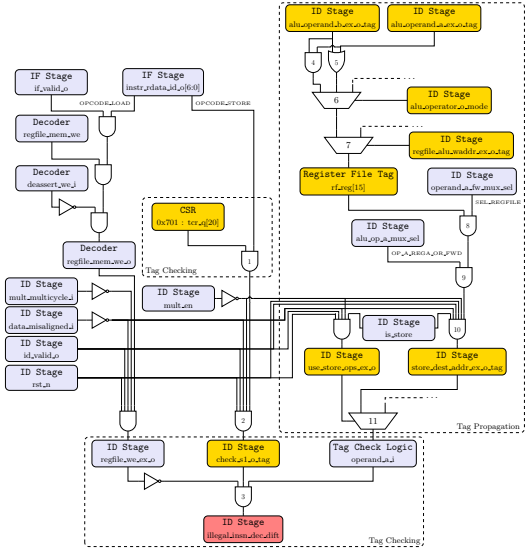


Figure 16: Logical analysis of the tags propagation in a *format string* attack

Case 3: Compare/Compute

- No software vulnerability
- Used to cover the DIFT surface

```
int main(){
    int a, b = 5, c;
    register int reg asm("x9");
    a = reg;
    asm volatile("csrw 0x700, tprValue");
    asm volatile("csrw 0x701, tcrValue");
    asm volatile("p.spsw x0, 0(\%0);" :: "r" (&a));
    c = (a > b) ? (a-b) : (a+b);
        //42c:   ble a4,a5,448
        //430:   addi a5,s0,-16
        //434:   lw a4,-12(a5)
        //438:   addi a3,s0,-16
        //43c:   lw a5,-4(a3)
        //440:   sub a5,a4,a5
        //444:   j 45c
        //448:   addi a5,s0,-16
        //44c:   lw a4,-12(a5)
        //450:   addi a3,s0,-16
        //454:   lw a5,-4(a3)
        //458:   add a5,a4,a5
        //45c:   sw a5,-24(s0)
    return EXIT_SUCCESS;
}
```

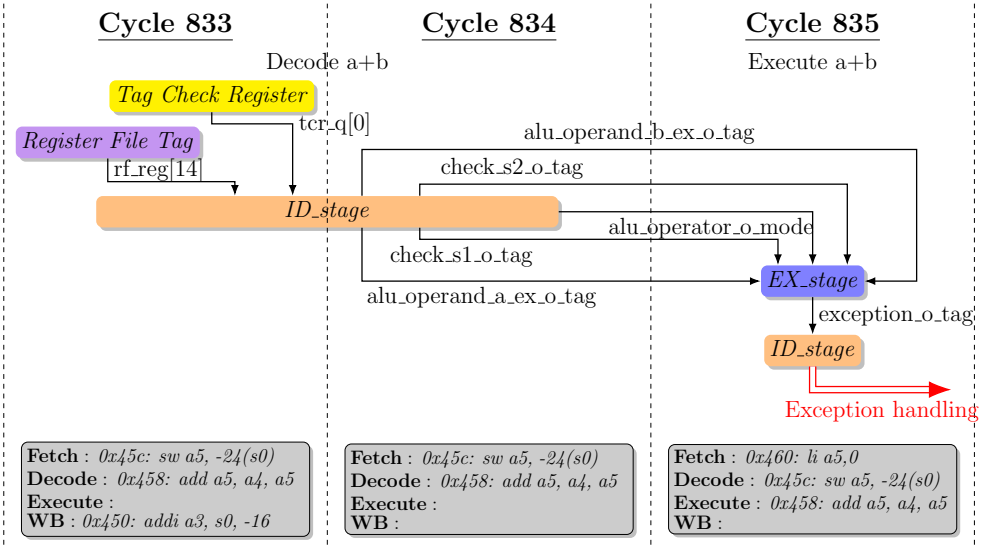


Figure 17: Temporal analysis of the tags propagation in a *format string* attack

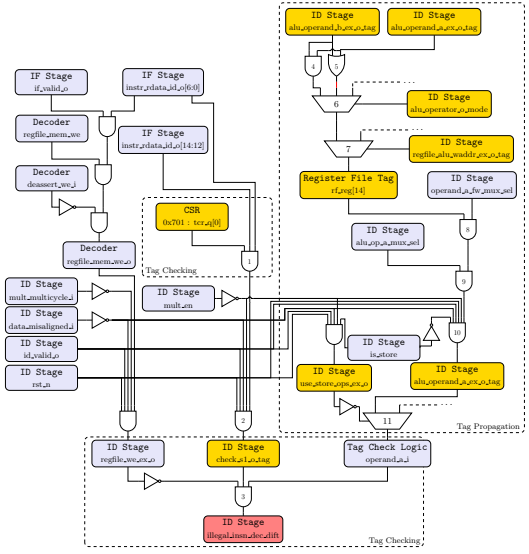


Figure 18: Logical analysis of the tags propagation in a *format string* attack