Attaque par Injection de Faute : Evaluation de Robustesse avec le Simulateur Embarqué

Séminaire sur la Confiance Numérique

Clermont-Ferrand

08 Janvier 2015

Lionel Rivière
lionel.riviere@morpho.com

PLAN

- **Introduction**
- **Concept du Simulateur de Faute Embarqué**
  - Intégration & mécanisme
- **Implémentation et Évaluations**
  - A. Expérience sur VerifyPIN
  - B. EFS vs. Simulation haut-niveau
- **Conclusion et Perspectives**

CADRE DES TRAVAUX

➔ Thèse CIFRE

- Morpho[1] — Télécom ParisTech[2]

➔ Directeurs

- Gérard COHEN
- Hervé CHABANNE

➔ Encadrant

- Thanh-Ha LE et Julien BRINGER

➔ Début de la thèse

- 1er Juillet 2012

Ces travaux sont issus de collaborations avec Maël Berthier[1], Julien Bringer[1], Hervé Chabanne[1,2], Thanh-Ha Le[1], Marie-Laure Potet[3], Maxime Puys[1] et Victor Servant[1].

Verimag[3]

/01/

Introduction

SAFRAN Morpho

INTRODUCTION
LES CARTES À PUCE

SAFRAN
Morpho

INTRODUCTION

LES CARTES À PUCE

Common Criteria

EMVCo

SAFRAN Morpho

LES CARTES À PUCE

Common Criteria

EMVCo

AGENCE NATIONALE DE LA SÉCURITÉ DES SYSTÈMES D'INFORMATION
ANSSI

Bundesamt
für Sicherheit in der
Informationstechnik

SAFRAN
Morpho

CARTE À PUCE ET MICROCONTRÔLEUR

➔Caractéristiques désirées

- Personnelle
- Peu coûteuse ←→Sécurisée

➔Sécurité

- Stockage d'information
- Mécanismes de sécurité
- Primitives cryptographique
- Confidentialité / Intégrité / Authenticité

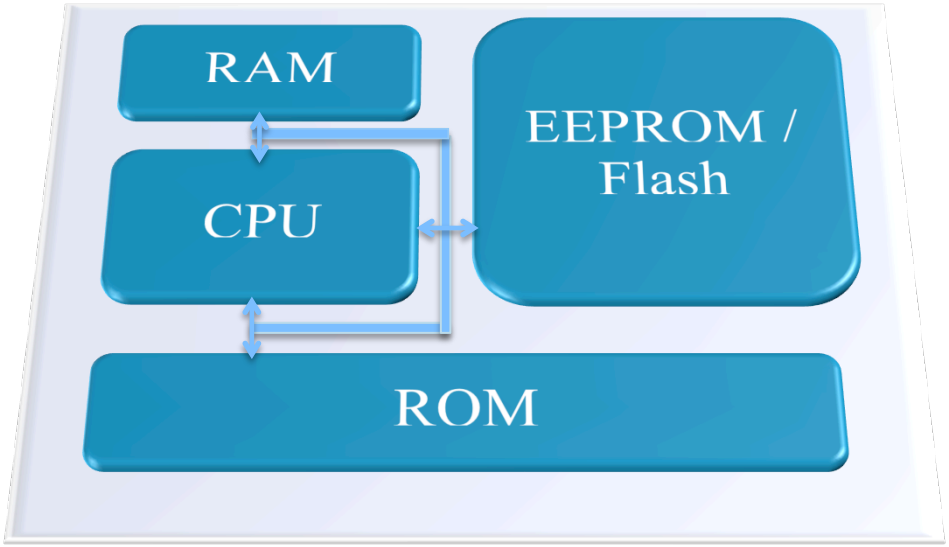
➔Architecture matérielle

- Processeur: CPU, crypto-processeur
- Mémoire : EEPROM, RAM, Flash
- Bus d'interconnexion

CARTE À PUCE ET MICROCONTRÔLEUR

➔Caractéristiques désirées

- Personnelle
- Peu coûteuse ←→Sécurisée



➔Sécurité

- Stockage d'information
- Mécanismes de sécurité
- Primitives cryptographique
- Confidentialité / Intégrité / Authenticité

➔Architecture matérielle

- Processeur: CPU, crypto-processeur
- Mémoire : EEPROM, RAM, Flash
- Bus d'interconnexion

CONTEXTE & BESOINS

➔Implémentations embarquées sujettes à des attaques spécifiques

- Attaque par canaux cachés
- Attaque par injection de faute

CONTEXTE & BESOINS

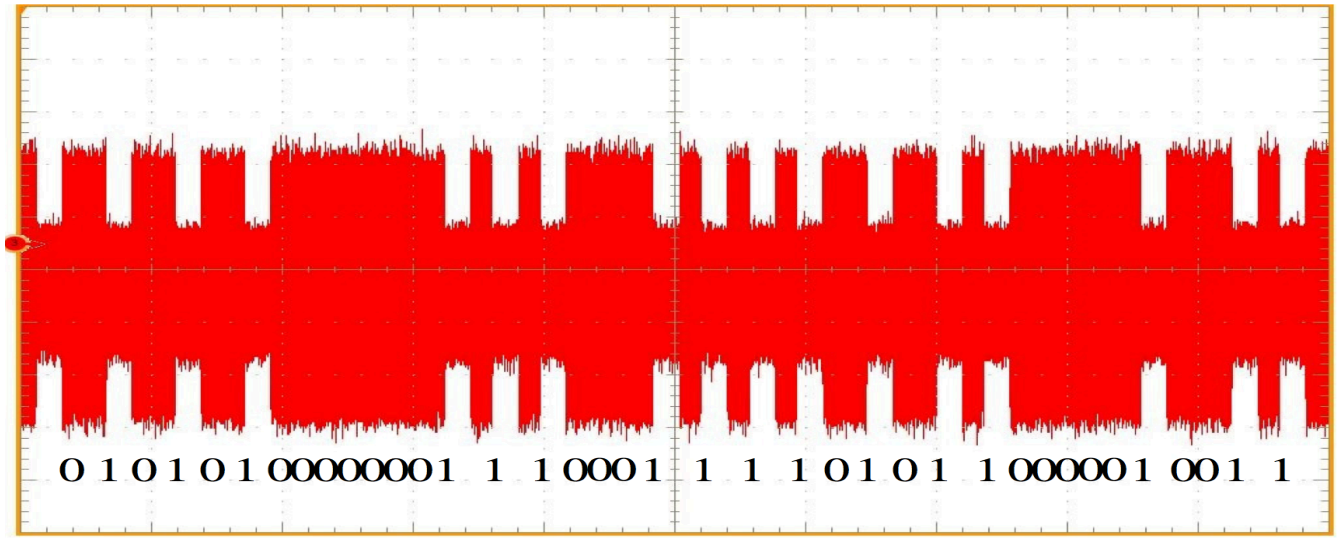
➔ Implémentations embarquées sujettes à des attaques spécifiques

- Attaque par canaux cachés
- Attaque par injection de faute

➔ Attaque par canaux cachés

- Mesure des grandeurs physiques : temps, consommation, radiation EM
- Dépendances entre une grandeur physique et des données manipulées

Exemple de l'exponentiation modulaire (square and multiply)



CONTEXTE & BESOINS

➔ Implémentations embarquées sujettes à des attaques spécifiques

- Attaque par canaux cachés
- **Attaque par injection de faute**

➔ Attaque physiques par perturbation

- Modification du comportement normal du composant
- But : Induire des erreurs exploitables
- Moyens : glitch, lumière, impulsion(s) laser, rayonnement EM

➔ Impacts

- Physique : porte/bascules, cellules mémoire, bus d'interconnexion …
- Logique :
  - Flot d'exécution → saut/remplacement d'instruction, inversion de test…
  - Donnée → bit flip, stuck-at-0/1 …

CONTEXTE & BESOINS

```
    79:     if (byteArrayCompare2(pin, buffer) == k_TRUE)
    80:     {
    81:         // Authentication();
3:0xF04870      MOV     WR22,#PIN(0x000D)
3:0xF04874      MOV     WR20,#C_STARTUP(0x0000)
3:0xF04878      MOV     WR18,#BUFFER(0x0013)
3:0xF0487C      MOV     WR16,#C_STARTUP(0x0000)
3:0xF04880      ECALL   BYTEARRAYCOMPARE2(0xF049A8)
3:0xF04884      CJNE    A,#0xAA,0xF048B3
```

- But : Induire des erreurs exploitables
- Moyens : glitch, lumière, impulsion(s) laser, rayonnement EM

➔ Impacts

- Physique : porte/bascules, cellules mémoire, bus d'interconnexion …
- **Logique** :
  - Flot d'exécution → saut/remplacement d'instruction, inversion de test…
  - Donnée → bit flip, stuck-at-0/1 …

Menace critique

INJECTION PHYSIQUE

➔Limitations expérimentales

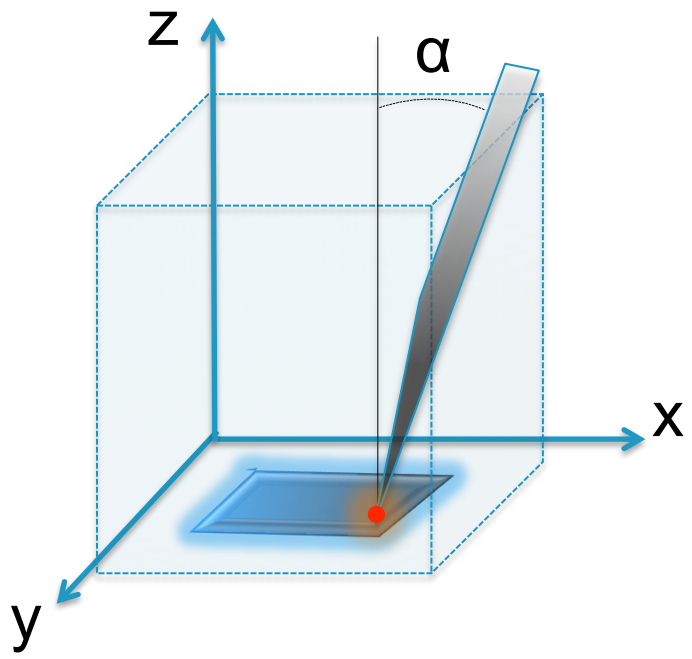
- Multiplicité des paramètres d'injections
  - Espace / Temps / Energie
- Couverture des paramètres

INJECTION PHYSIQUE

➔ Limitations expérimentales

- Multiplicité des paramètres d'injections
  - **Espace** / Temps / Energie
- Couverture des paramètres

→ (x,y,z,α) surface du point d'injection, taille du pas

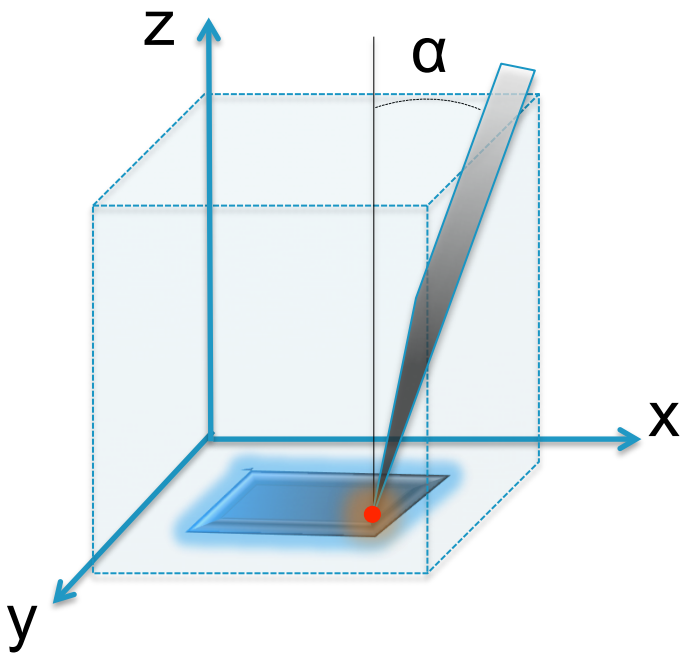


INJECTION PHYSIQUE

➔Limitations expérimentales

- Multiplicité des paramètres d'injections
  - Espace / **Temps** / Energie
- Couverture des paramètres

→ Déclenchement d'injection, délais, durée, temps de recharge

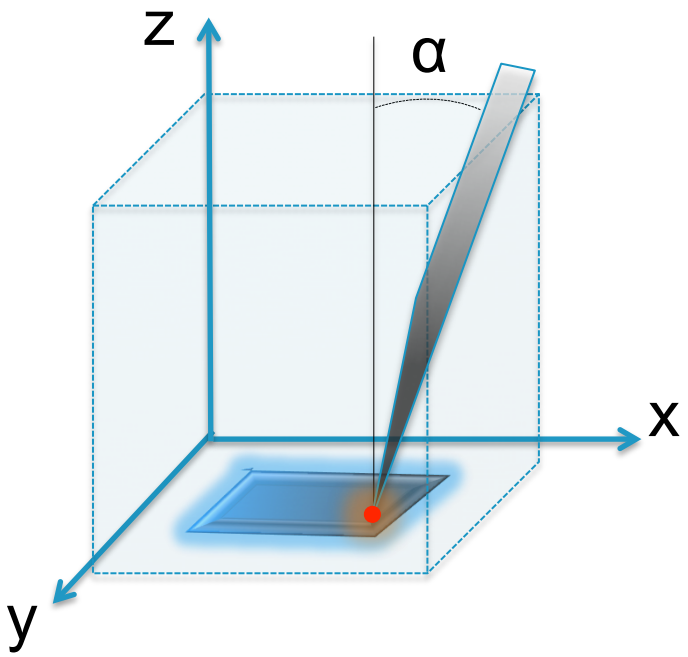


INJECTION PHYSIQUE

➔Limitations expérimentales

- Multiplicité des paramètres d'injections
  - Espace / Temps / **Energie**
- Couverture des paramètres

→ Longueur d'onde, forme d'onde EM, intensité, puissance, phase, température



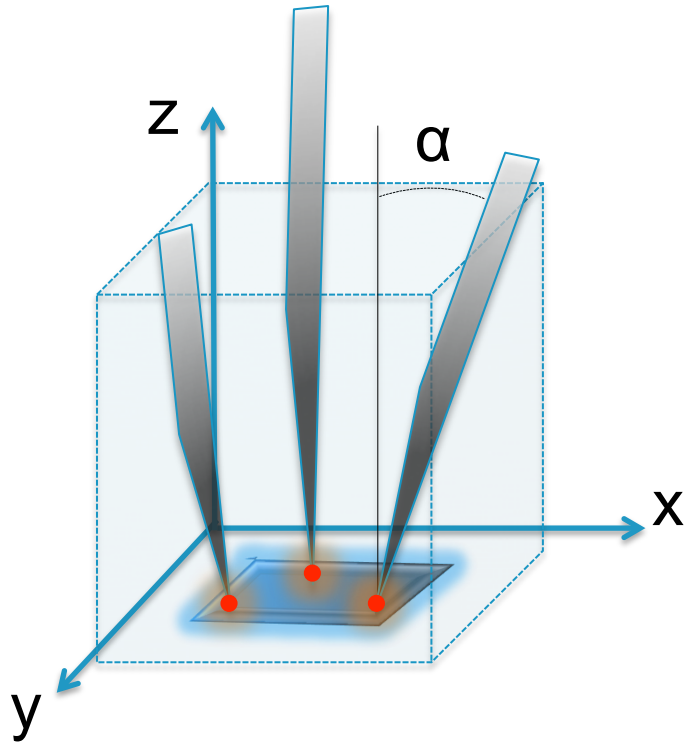
INJECTION PHYSIQUE

➔ Limitations expérimentales

- Multiplicité des paramètres d'injections
  - Espace / Temps / Energie
- Couverture des paramètres

➔ Injections multiples

- Explosion de la combinatoire
- Complexité du setup



QUELLES SOLUTIONS ?

➔Injection physique de faute

- Couverture (espace / temps / énergie)
- Explosion combinatoire en faute multiple
- Coût et complexité des plateformes

➔Simulation de faute

➔Contexte de développeur carte à puce

- Une seule vulnérabilité exploitée peut remettre en question tout un produit
- Contrainte de temps et coûts

BESOINS ET RECOMMANDATIONS

➔Environnement

- Morpho : développeur logiciel sécurisé
- Accès à une large gamme de CàP basés sur des composants variés
- Aucun contrôle sur le matériel (pas de modification matérielle possible)

➔Besoin: Logiciel d'injection de faute

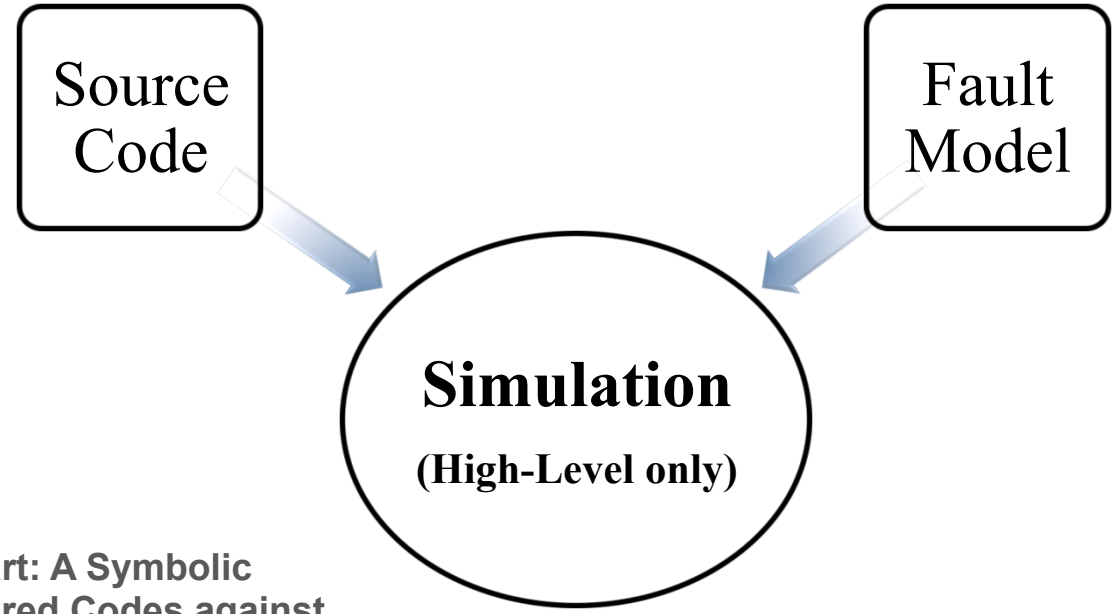
- Automatisé pour de la recherche exhaustive de vulnérabilité
- Compatible avec le maximum d'architecture CàP
- Compatible avec les différents OS
- Capacité de fauter les données et le flot de contrôle
- Possibilité de fautes multiples

Simulateur de Faute Embarqué (EFS)

SOLUTIONS EXISTANTES

➔Simulation haut-niveau

- Pas de considérations matérielles
- Modèle de faute complexe
- Très performant



M.-L. Potet, L. Mounier, M. Puys et L. Dureuil – **Lazart: A Symbolic Approach for Evaluation the Robustness of Secured Codes against Control Flow Injections** – ICST 2014

J.-F. Lalande, K. Heydemann et P. Berthomé – **Software countermeasures for control flow integrity of smart card C code** – ESORICS 2014

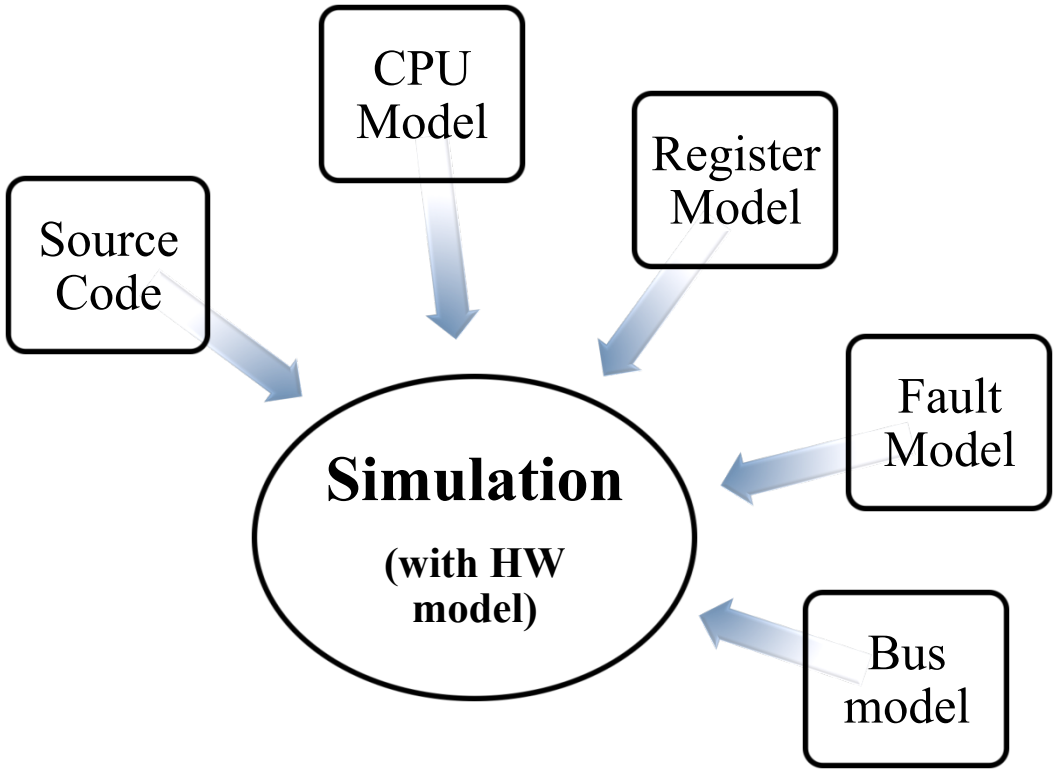
SOLUTIONS EXISTANTES

➔ **Simulation haut-niveau**

- Pas de considérations matérielles
- Modèle de faute complexe
- Très performant

➔ **Simulation haut-niveau avec considération matérielle**

- Complexe
- Spécifique
- Lent



P. Berthomé, *et al* - High Level Model of Control Flow Attacks for Smart Card Functional Security. In ARES 2012

SOLUTIONS EXISTANTES

Simulation haut-niveau	Généricité	Taux d'injection	Réalisme
Sans considération matérielle			
Avec considération matérielle			

➔ **Le réalisme de la simulation à haut-niveau dépends du modèles de faute.**

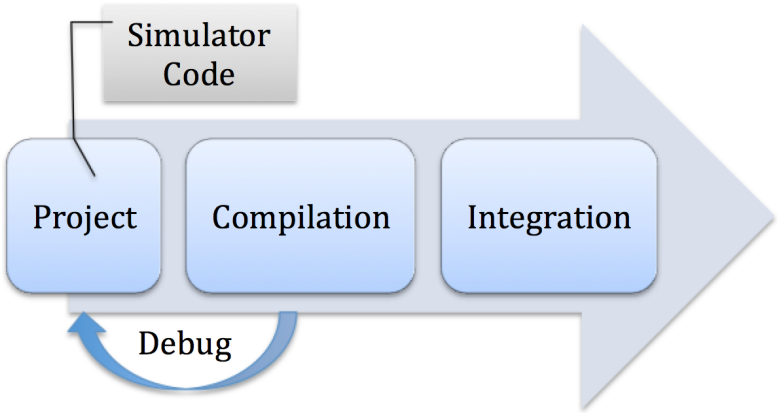
/02/

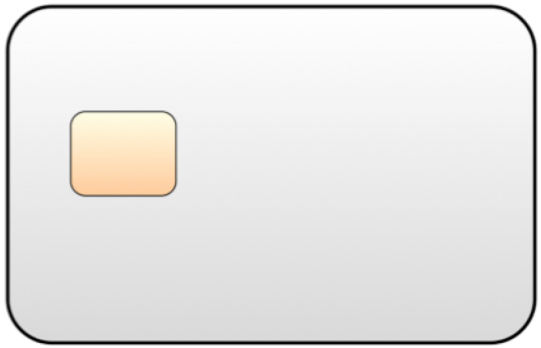
Le concept du Simulateur de Faute Embarqué *(EFS en anglais)*

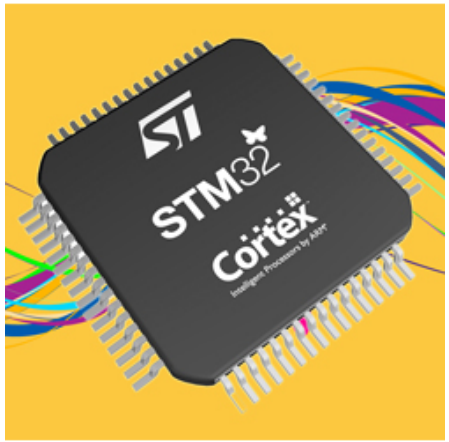
SAFRAN Morpho

INTÉGRATION

➔Intégration Projet







➔Paramètres via APDU sur carte via fonctions sur microcontrôleurs

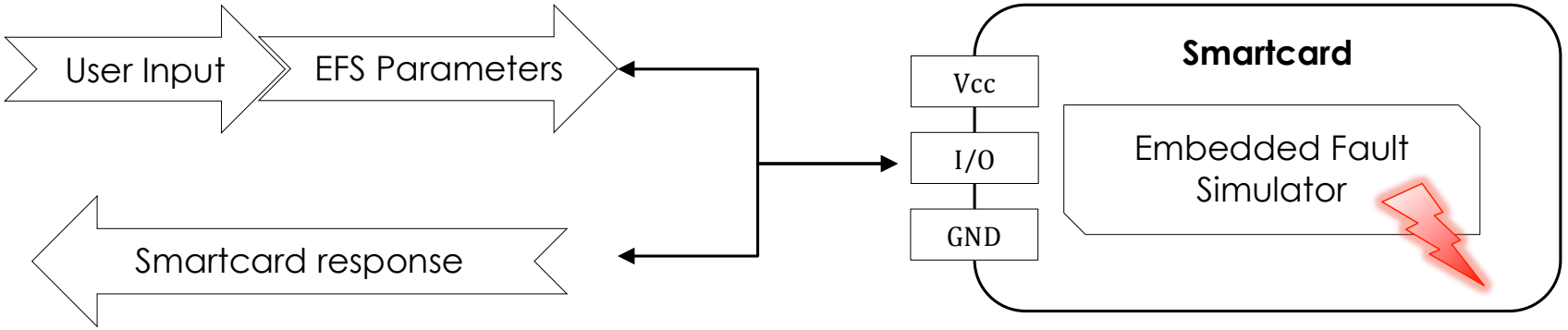
Parameter	Name	Size (in byte)	Example (hex.)
Identifier	id	2	C1 3E
Counter	cpt	1	01
Delay	delay	2	01 F5
Offset	offset	1	03
Increment	incr	1	08

➔Microcontrôleur

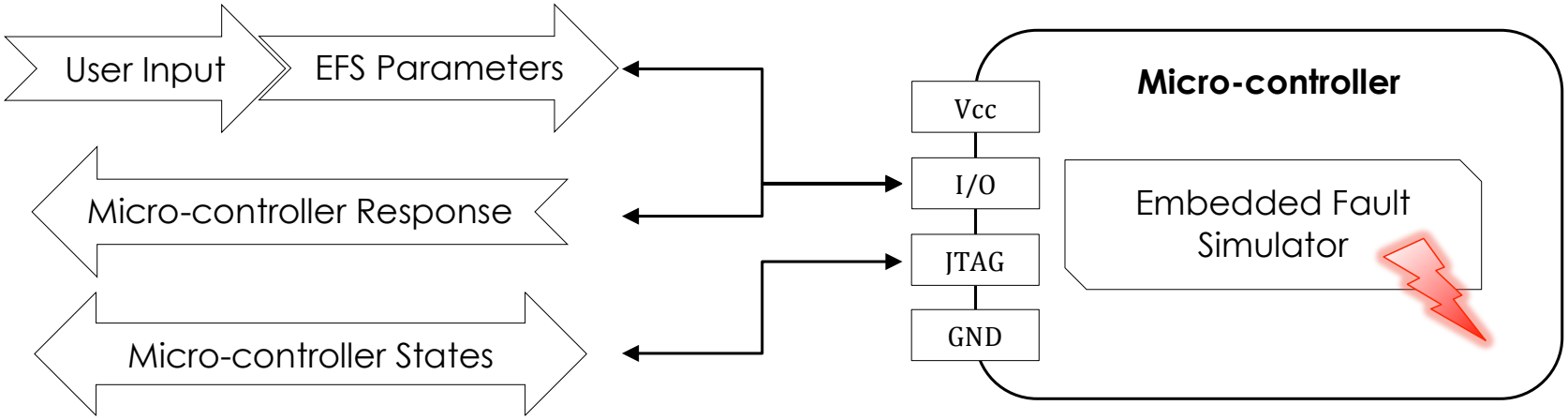
- Fonction cible
- Modèle de faute

DIAGRAMME FONCTIONNEL

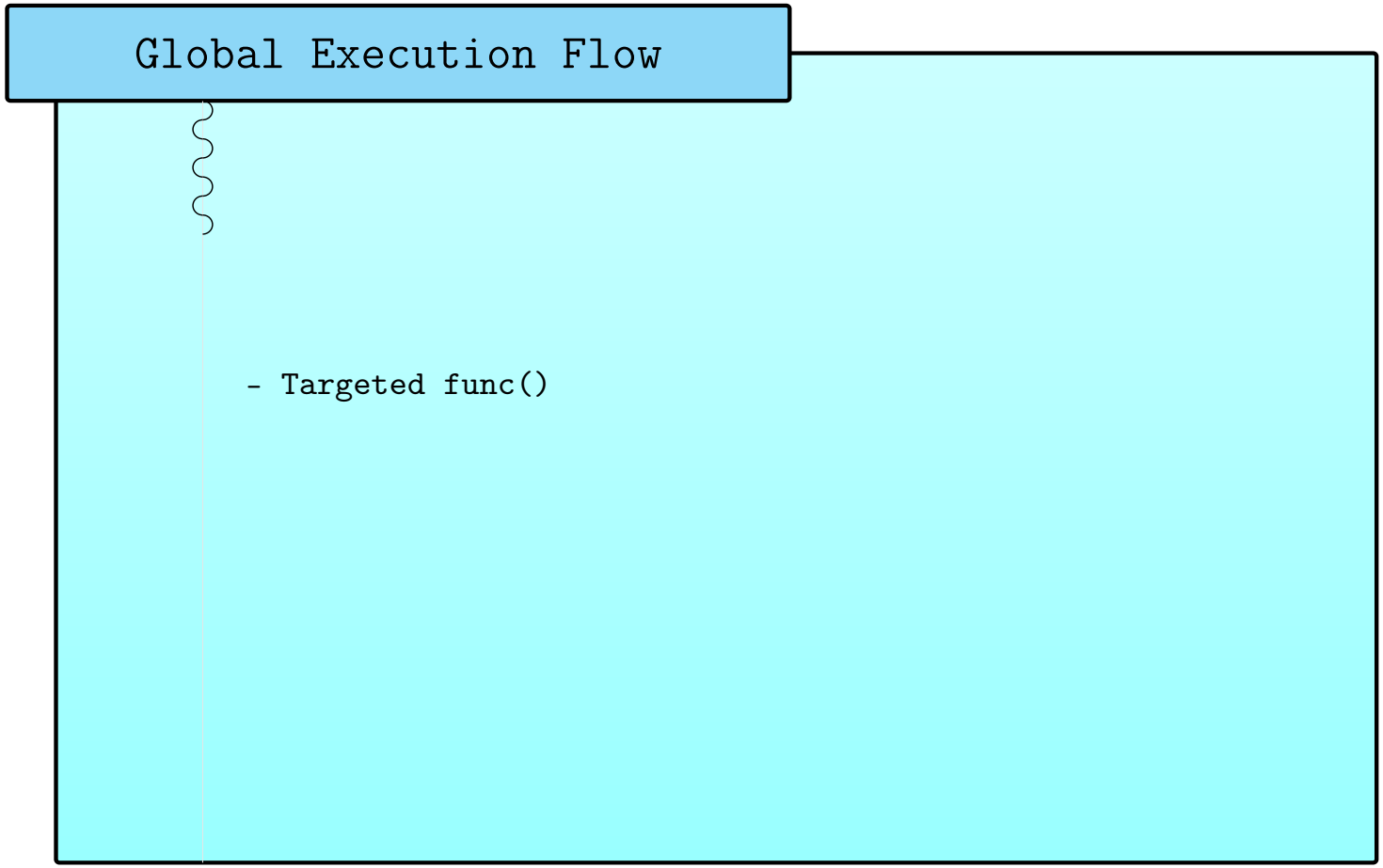
➔EFS sur Carte à Puce



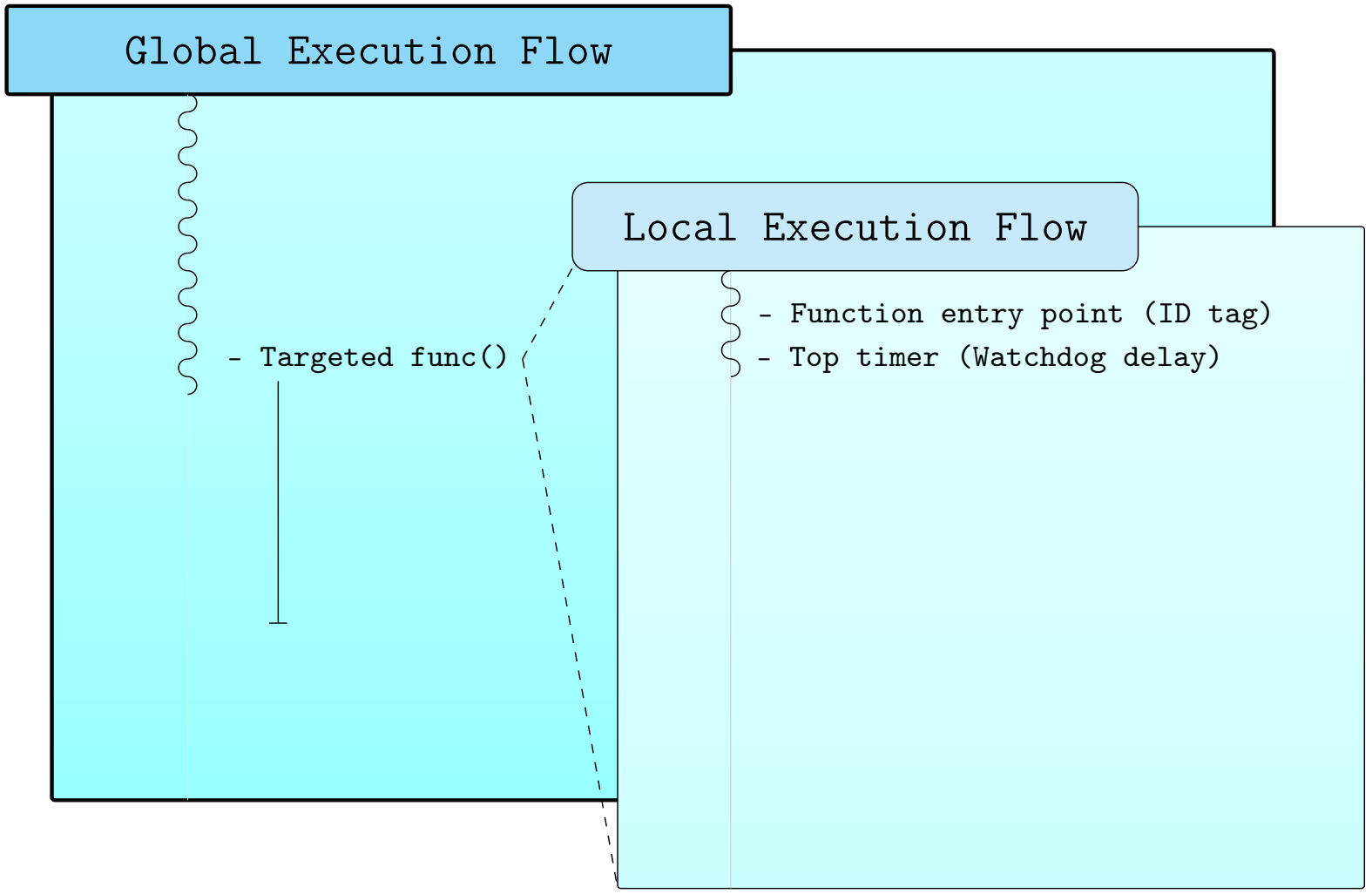
➔EFS sur Microcontrôleur



MÉCANISME D'INJECTION

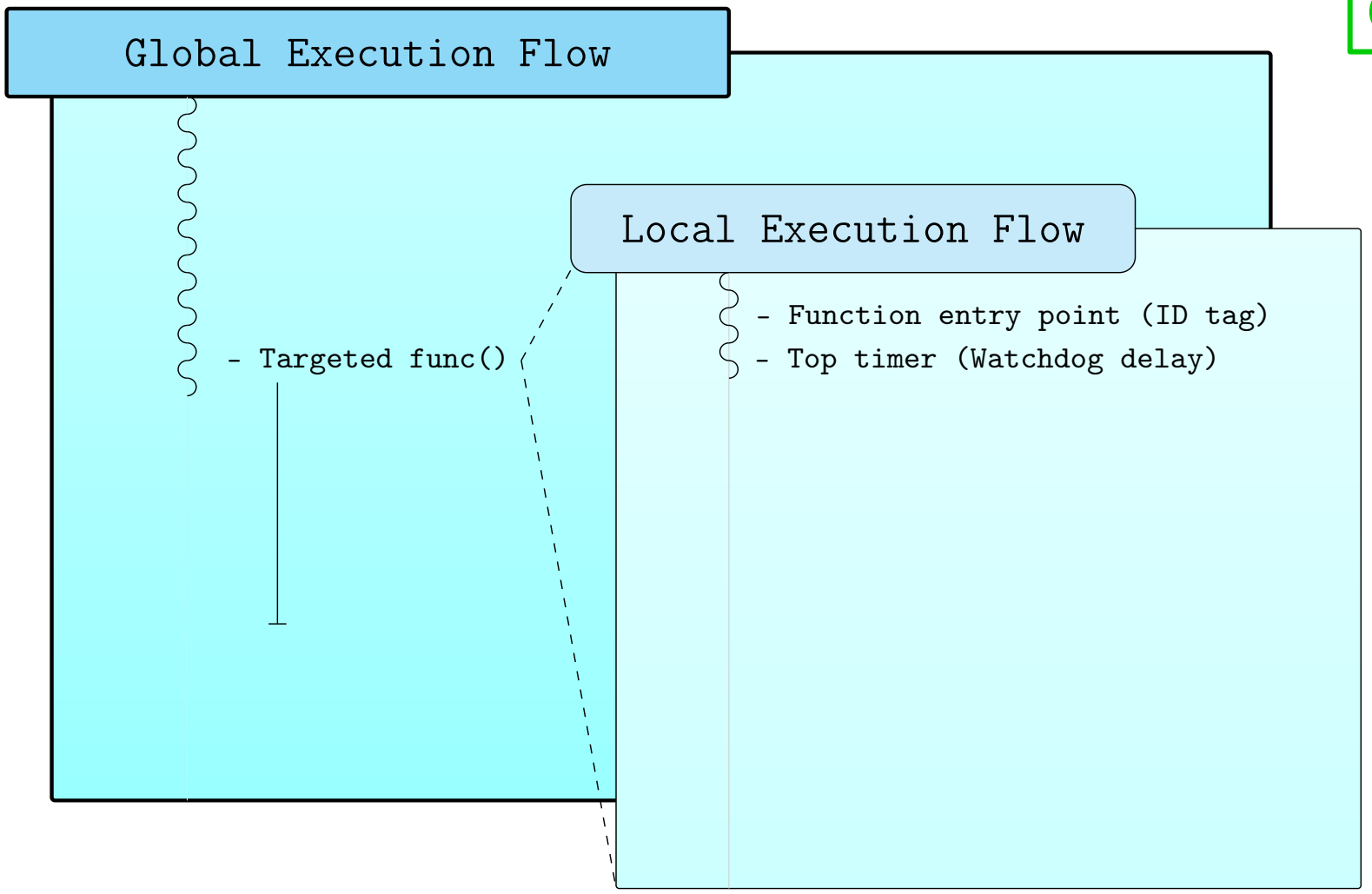


MÉCANISME D'INJECTION



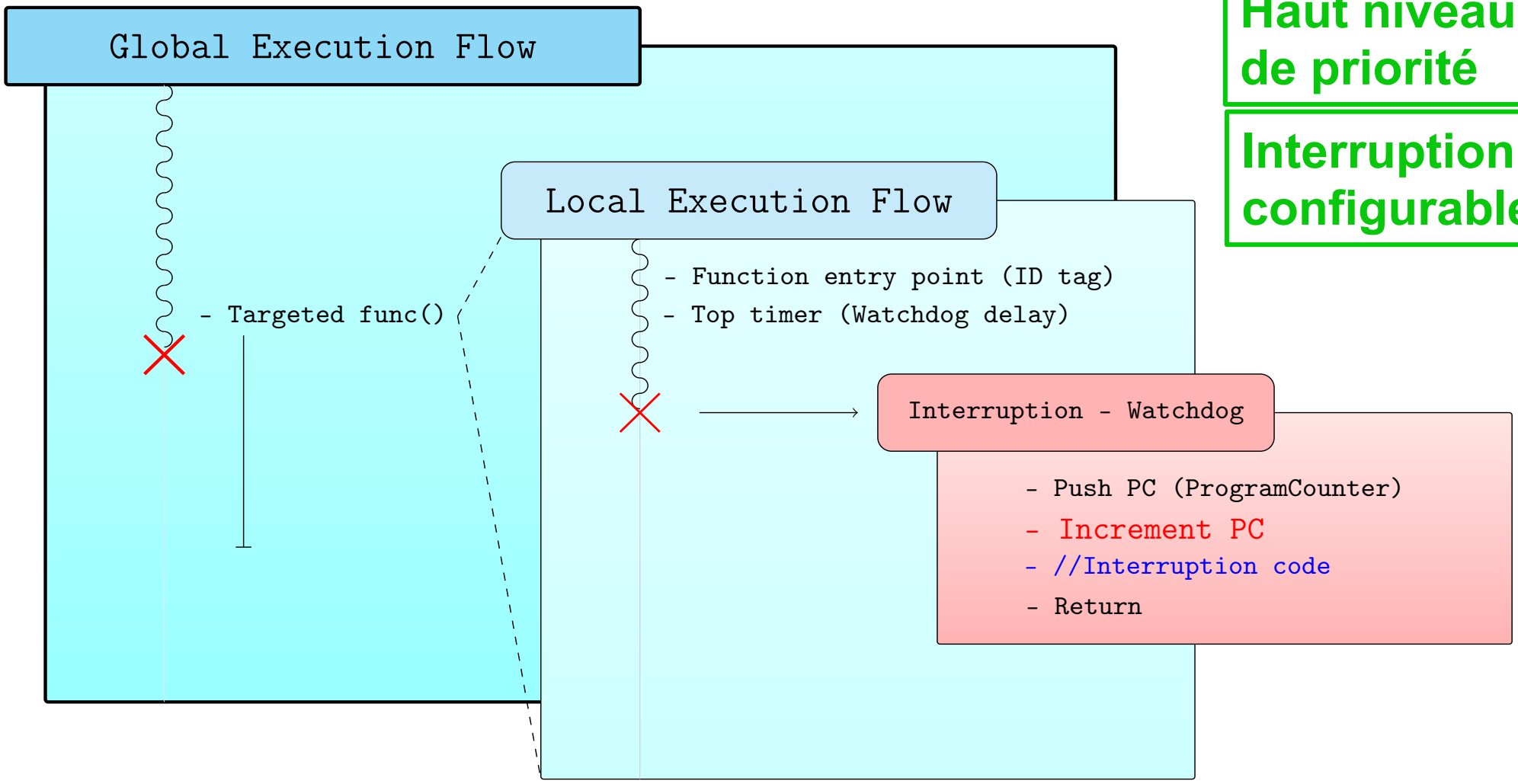
MÉCANISME D'INJECTION

Ciblage précis



SAFRAN Morpho

MÉCANISME D'INJECTION

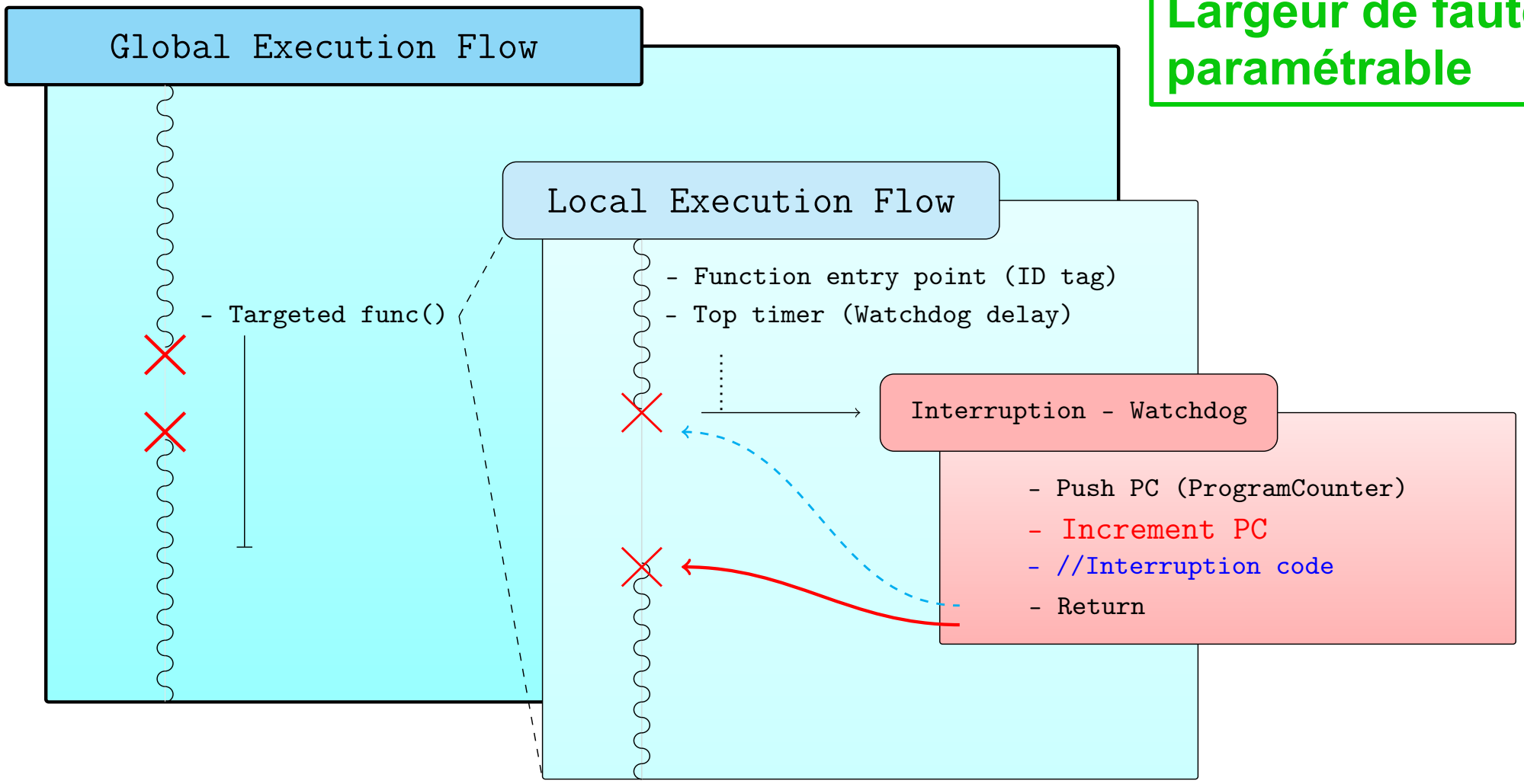


Haut niveau de priorité

Interruption configurable

MÉCANISME D'INJECTION

Largeur de faute paramétrable

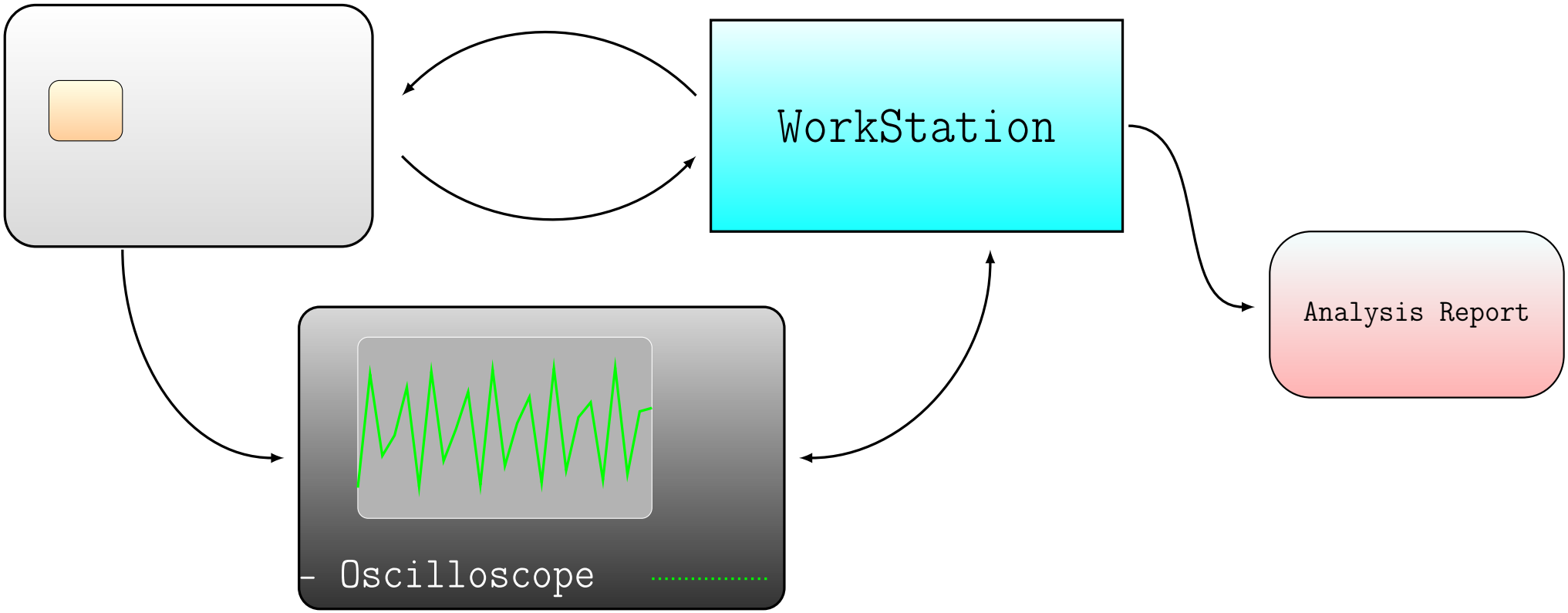


SAFRAN Morpho

SIMULATEUR DE FAUTE EMBARQUÉ

BANC DE TEST + SIDE CHANNEL

Observation Side-Channel



SAFRAN Morpho

MODÈLES DE FAUTE CONNUS

➔Effet de faute considéré

bit : b					byte : B		
set	flip	rst	stuck	rand	set	rst	rand
↓	↓	↓	↓	↓	↓	↓	↓
1	$\neg b$	0	b	{0,1}	0xFF	0x00	$\{0,1\}^8$

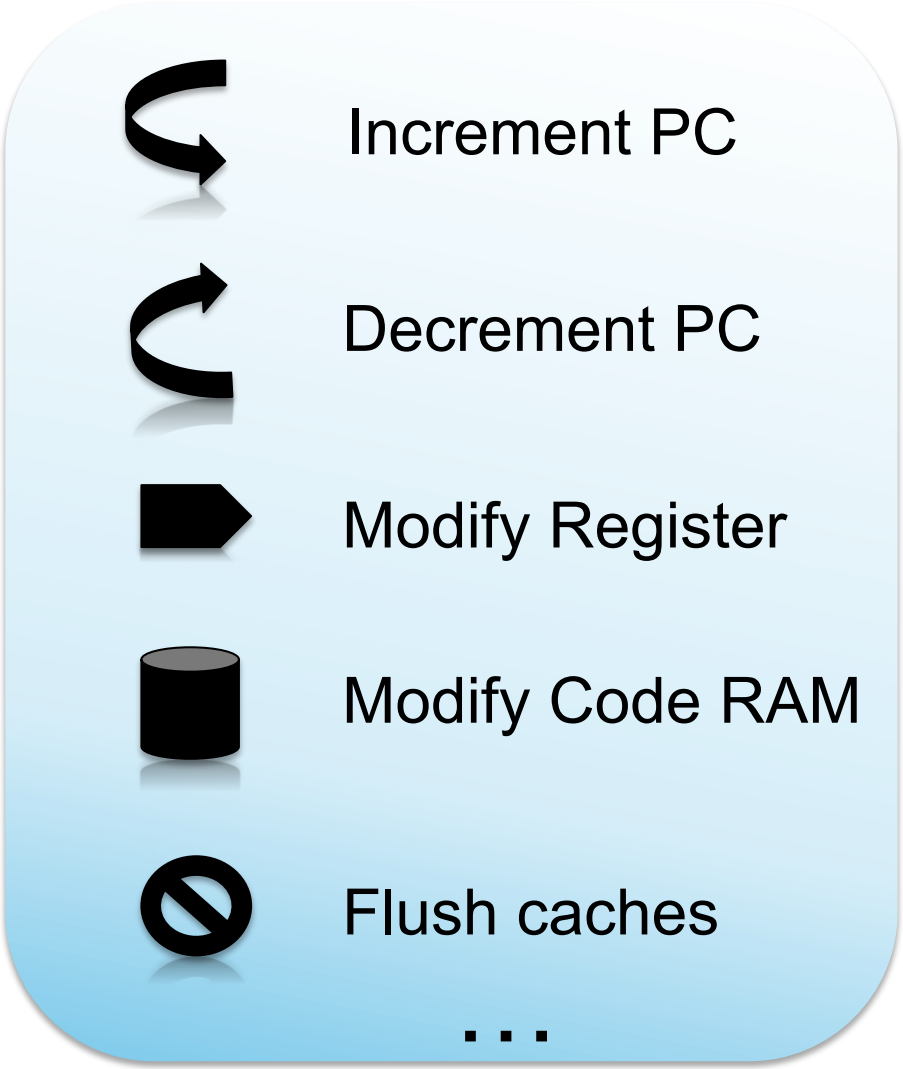
➔Conséquence sur le flot de control

- Modification d'instruction
- Remplacement d'instruction

➔Conséquence sur les données

- Modification de valeur
- Mise à zéro de valeur

MODÈLES D'ATTAQUANT



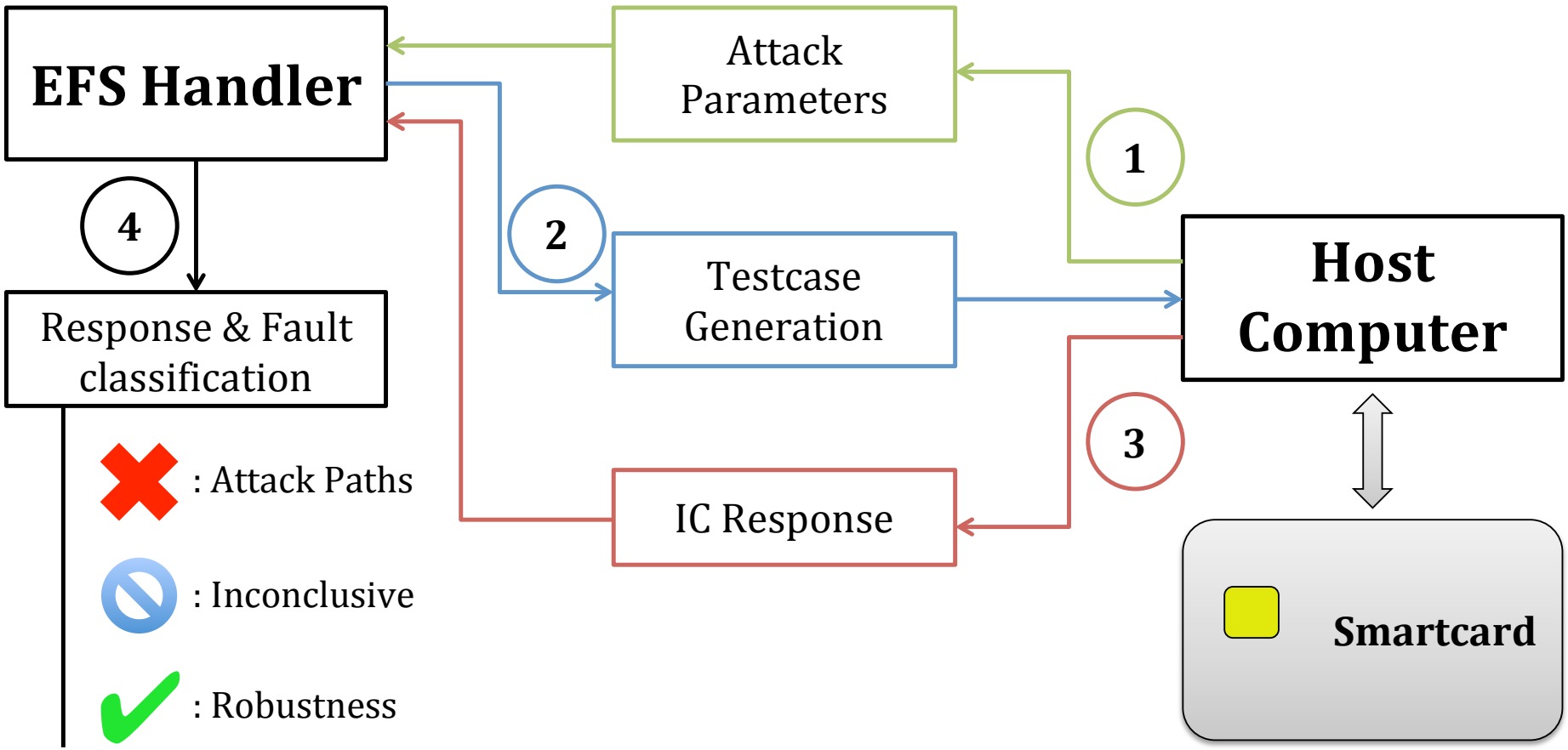
➔Code de l'interruption programmable

- Contexte d'exécution de la fonction interrompue dans la pile
- Registres généraux bas accessibles
- Registres spéciaux accessibles

➔Implémentation libre

- Le programmeur peut choisir le code d'interruption à exécuter

SIMULATEUR DE FAUTE EMBARQUÉ
DIAGRAMME FONTIONNEL



SAFRAN Morpho

PERFORMANCES

➔ Attaque Exhaustive de fonctionnalités complètes avec branchements (en 2013)

- GetSecurityAttribute – 8 Instructions / 24 cycles
- AES – ~600 instructions / ~2 000 cycles

		Faute simple		Faute double	
	Approche	largeur fixe 1	largeur variable	largeur fixe 1	largeur variable
8 instr. 24 cycles	Connaissance du code	~ 15 s	< 3 m	< 3 m	< 40 m
	Naïve	~ 25 s	< 6 m	< 5 m	< 8 H
~600 instr. 2 000 cycles	Connaissance du code	~ 25 m	< 7 jours	< 15 j	>1 an
	Naïve	~ 30 m	< 28 jours	< 20 j	>1 an

➔ Attaque Exhaustive de fonctionnalités complète avec branchements (en 2014)

- VerifyPIN protégé – 800 instructions / ~2300 cycles

	Faute simple		Faute double	
Approche	$\mathcal{F}_w = 1$	$\mathcal{F}_w = 1$ to 5	$\mathcal{F}_w = 1$	$\mathcal{F}_w = 1$ to 5
Sans échec	49 s	3,9 m	4,1 m	22 m
Avec échec	50 m	1,45 h	>4 h	>12 h

PERFORMANCES

➔ Attaque Exhaustive de fonctionnalités complètes avec branchements (en 2013)

- GetSecurityAttribute – 8 Instructions / 24 cycles
- AES – ~600 instructions / ~2 000 cycles

		Faute simple		Faute double	
	Approche	largeur fixe 1	largeur variable	largeur fixe 1	largeur variable
8 instr. 24 cycles	Connaissance du code	~ 15 s	< 3 m	< 3 m	< 40 m
	Naïve	~ 25 s	< 6 m	< 5 m	< 8 H
~600 instr. 2 000 cycles	Connaissance du code	~ 25 m	< 7 jours	< 15 j	>1 an
	Naïve	~ 30 m	< 28 jours	< 20 j	>1 an

➔ Attaque Exhaustive de fonctionnalités complète avec branchements (en 2014)

- VerifyPIN protégé – 800 instructions / ~2300 cycles

Jusqu'à 95% de temps gagné en simulation

	Faute simple		Faute double	
Approche	$\mathcal{F}_w = 1$	$\mathcal{F}_w = 1$ to 5	$\mathcal{F}_w = 1$	$\mathcal{F}_w = 1$ to 5
Sans échec	49 s	3,9 m	4,1 m	22 m
Avec échec	50 m	1,45 h	>4 h	>12 h

RÉSUMÉ DES CARACTÉRISTIQUES DU SIMULATEUR

➔Mécanisme de simulation d'injection

- Ciblage fin
- Haut niveau de priorité en exécution dynamique
- Propriété du modèle de faute ajustable (largeur, permanente, transitoire)

➔Capacités

- Impact sur le flot de contrôle et les données
- Approche et spécifications génériques
- Code haut niveau portable / développement spécifique minimal à bas niveau
- Possibilité d'observations Side-Channel dans le même temps

➔Intérêt

- Trouver toutes les déviations fonctionnelles par rapport aux spécifications
- Analyse niveau assembleur, plus proche du matériel

/03/

Implémentation et Évaluation du Simulateur Embarqué

A – Expériences sur VerifyPIN

EFS – Expérience sur VerifyPIN
BUT

➔Sujet

- Etude de scénarios d'attaque sur 2 implémentations simples

➔Objectif & Intérêt

- Valider l'intégrité du PTC quelque soit le point d'entrée en faute
- Efficacité des tests d'intégrité (utilité des tests redondants : copie et recalcule)
- Impact du mode de développement inter-procédural

➔Expériences réalisées

- EFS sur VerifyPIN Call (inter-procédural)
- EFS sur VerifyPIN Local (monolithique)
- Acquisitions fonctionnelles
- Réponses APDU pour CàP <> log texte pour microcontrôleur

BANC DE TEST

➔PC de développement

- Programme en C sous l'IDE : Keil

➔PC de tests

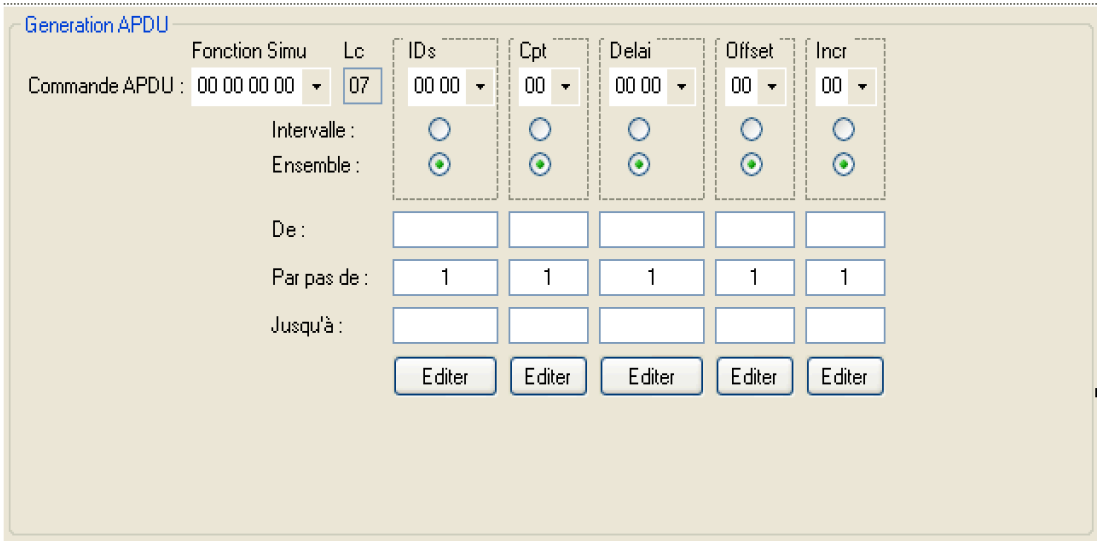
- Génération automatique des chemins d'attaques
- Module d'acquisition fonctionnelle

➔Cartes & Lecteur

- Microcontrôleur 16-bits
- CardMan 3x21 PCSC
- Protocole T0

➔Board STM32 F4

- Interfaçage USB



CIBLE 1 : VERIFYPIN

```
    if (byteArrayCompare2(cardPinValue, userPinValue))
    {
        // Authentication();
        pinTryCounter++;

        // PTC Verification 4 : Check if the incrementation occured
        pinTryCounterCpy++;
        memWrite((byte*)&pinTryCounter, ptcAddress, sizeof(byte));

        // PTC Verification 5 : Retrieve what we just wrote in NVM to check its value
        status = true;
    }
    else
    {
        // Mutecard();
        abort();
    }
}
else
{
    // Balanced branch
```

Threat	Resistance
Card Tearing	$\surd$
Side-Channel leakage	$\surd$
Instruction skipping	Δ (Double Check)
Test Inverting	Δ
Data modification	Δ (PTC)

$\surd$: Yes $\times$: No Δ : Partially

CIBLE 2 : VERIFYPIN LOCAL

```
// Main Comparison
equal = BOOL_TRUE;
for(i=0 ; i<SIZE_OF_PIN; i++)
{
    equal = equal & ((buffer[i] != pin[i]) ? BOOL_FALSE : BOOL_TRUE);
    stepCounter++;
}

if(equal == BOOL_TRUE)
{
    // Comparison is successful
    if(equal != BOOL_TRUE)
        goto counter_measure;

    // Resets the remaining tries to the max
    triesLeft = MAX_TRIES;
    triesLeftBackup = -MAX_TRIES;

    // Verifies the new value
    if(triesLeft != -triesLeftBackup)
        goto counter_measure;

    authenticated = 1;

    if(stepCounter == INITIAL_VALUE + 4)
        return EXIT_SUCCESS;

}
else
{
```

Threat	Resistance
Card Tearing	Δ (Increment comes after comparison)
Side-Channel leakage	$\times$
Instruction skipping	Δ (Inverse condition)
Test Inverting	$\surd$
Data modification	Δ (PTC)

$\surd$: Yes $\times$: No Δ : Partially

EFS – EXPÉRIENCE SUR VERIFYPIN
CIBLES

➔Comparaison des propriétés des cibles

	Fonction	
Propriété	VerifyPIN Local	VerifyPIN Call
Nb ligne C	83	129
NB instructions C	36	53
Comp. tps. constant	✔	✔
Double comp.	✔	✔
Profondeur de l'authentification	2	4
Equilibrage des branches conditionnelles	✖	✔

EFS – EXPÉRIENCE SUR VERIFYPIN
MÉTHODOLOGIE

➔Phase 1 : Attaque exhaustive

- Seul l'aspect de terminaison fonctionnelle est considéré
- Sur toute la durée de la fonction cible
- Paramètres variant de l'attaque
  - Délais de déclanchement du simulateur
  - Largeur de faute

➔Phase 2 : Attaque de tous les chemins non-bloquants déterminés en phase 1

- Génération de log
- Sauvegarde des paramètres du simulateur
- Localisation dans le code

EFS – EXPÉRIENCE SUR VERIFYPIN
SCENARIO

➔Exécution en boucle avec PIN Faux

- Plage de déclanchement du simulateur : Début – Fin fonction
- Précision : A chaque instruction
- Largeur de faute
  - Cas 1 : de 0 à 22 octets (VerifyPIN Local)
  - Cas 2 : de 0 à 128 octets (VerifyPIN Call)

➔Nombre de test

- Cas 1 : 4 540
- Cas 2 : 40 770

00 10 01 11 07 01 3D 00 **00 0D 00 03**	55 90 00	F0 B2 C2 F0 00 3B F0 01 BF 00 00 F0 0F 1F FF 02 45 F0 48 14 **03 F0 B1 CE** 90 00
00 10 01 11 07 01 3D 00 **00 0D 00 09**	6F FC	F0 B2 C2 F0 00 3B F0 01 BF 00 00 F0 0F 1F FF 02 45 F0 48 14 **03 F0 B1 CE** 90 00
00 10 01 11 07 01 3D 00 **00 0D 00 0B**	6F FE	F0 B2 C2 F0 00 3B F0 01 BF 00 00 F0 0F 1F FF 02 45 F0 48 14 **03 F0 B1 CE** 90 00
00 10 01 11 07 01 3D 00 **00 0E 01 5C**	01 90 00	CA 8F 7F F0 B2 C2 F0 00 3B F0 01 BF 00 00 F0 0F 1F FF 02 45 **03 F0 48 14** 90 00
00 10 01 11 07 C1 3E 00 **00 0C 03 00**	APDU error	APDU error

SCENARIO

➔ Exécution en boucle avec PIN Faux

- Plage de déclanchement du simulateur : Début – Fin fonction
- Précision : A chaque instruction
- Largeur de faute
  - Cas 1 : de 0 à 22 octets (VerifyPIN Local)
  - Cas 2 : de 0 à 128 octets (VerifyPIN Call)

➔ Nombre de test

- Cas 1 : 4 540
- Cas 2 : 40 770

Adresses dans le code

Increment

Délai

00 10 01 11 07 01 3D 00	**00 0D**	**00**	**03**	55 90 00	F0 B2 C2 F0 00 3B F0 01 BF 00 00 F0 0F 1F FF 02 45 F0 48 14	**03 F0 B1 CE**	90 00
00 10 01 11 07 01 3D 00	**00 0D**	**00**	**09**	6F FC	F0 B2 C2 F0 00 3B F0 01 BF 00 00 F0 0F 1F FF 02 45 F0 48 14	**03 F0 B1 CE**	90 00
00 10 01 11 07 01 3D 00	**00 0D**	**00**	**0B**	6F FE	F0 B2 C2 F0 00 3B F0 01 BF 00 00 F0 0F 1F FF 02 45 F0 48 14	**03 F0 B1 CE**	90 00
00 10 01 11 07 01 3D 00	**00 0E**	**01**	**5C**	01 90 00	CA 8F 7F F0 B2 C2 F0 00 3B F0 01 BF 00 00 F0 0F 1F FF 02 45	**03 F0 48 14**	90 00
00 10 01 11 07 C1 3E 00	**00 0C**	**03**	**00**	APDU error	APDU error		

SAFRAN Morpho

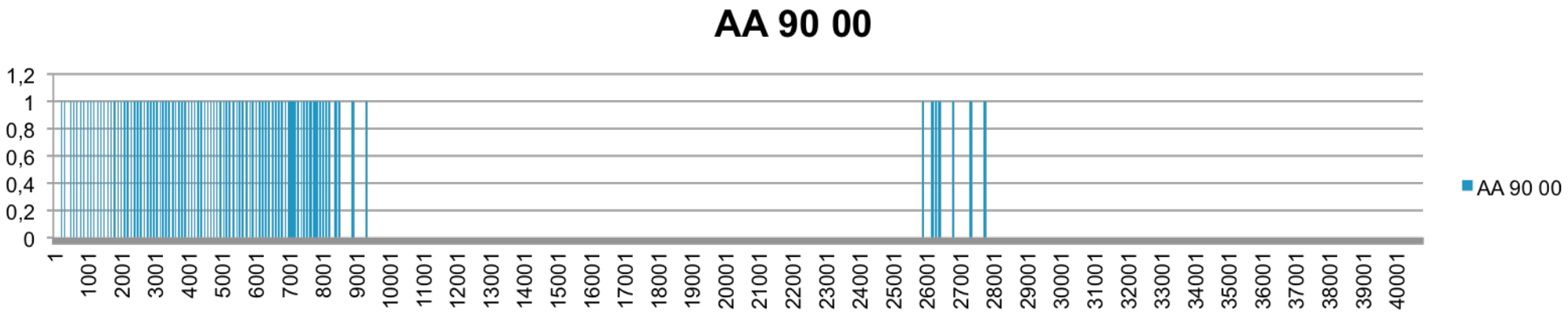
EFS – EXPÉRIENCE SUR VERIFYPIN
SORTIES FONCTIONNELLES

➔Comportements observés

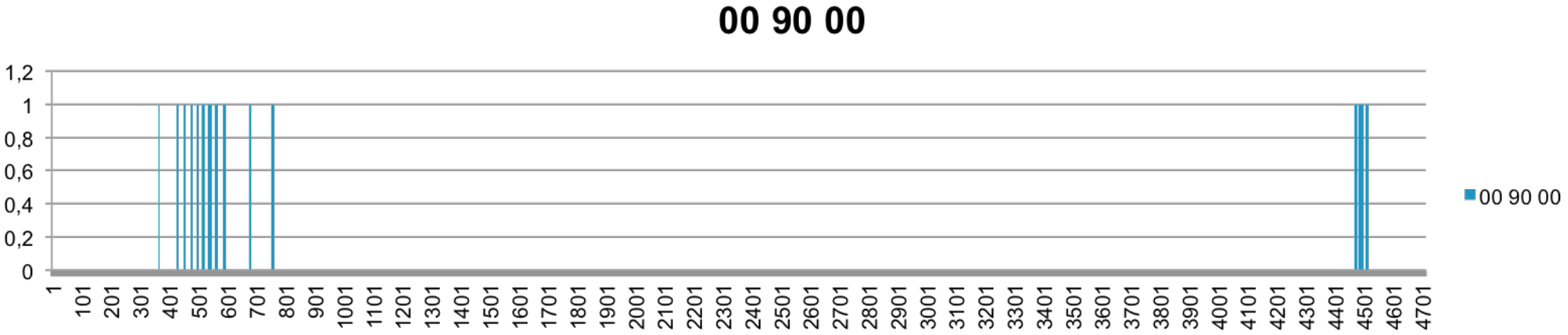
Sortie fonctionnelle	**Ratio** (sortie fct / nb total)	
	VerifyPIN Local	**VerifyPIN Call**
CTM	35,66%	44,70%
PIN Faux	32,93%	43,17%
Mute	11,28%	10,49%
PIN Vrai	**1,59%**	**1,41%**
Sortie aléatoire	17,64%	0,19%
Effacement et destruction	0,55%	0,03%
Erreur APDU	0,35%	< 0,01%
	100 %	**100 %**

EFS – EXPÉRIENCE SUR VERIFYPIN
ZONES DE SENSIBILITÉ

➔Fenêtre temporelle – VerifyPIN Call



➔Fenêtre temporelle – VerifyPIN Local



SAFRAN Morpho

COMPARATIF DES RÉSULTATS - CONCLUSION

➔Résultats haut-niveau avec Lazart

Vulnérabilité découverte par VERIMAG	3	3
- Attaque sur PIN faux	2 doubles	2 doubles
Détails	*1) l.301 + l.303* *2) l.120 2 fois de suite*	*1) l.44 + l.66* *2) l.49 + l.51*
- Compteurs	1 simple	1 simple
Détails	*l.271*	*l.28*

➔Largeur de saut minimale effective

- 7 octets – VerifyPIN Call
- 4 octets – VerifyPIN Local
- Certains résultats en faute double trouvés par Lazart sont obtenus en 1 seul saut long (>10 octets d'opcode) avec l'EFS

COMPARATIF DES RÉSULTATS - CONCLUSION

➔Rappel des objectifs

- Intégrité du PTC valide quelque soit le point d'entrée?
  - Oui pour les deux implémentations
- Efficacité des tests d'intégrité
  - utilité des tests redondants : copie et recalcule
  - Sans redondance, les fautes simples permettent de déjouer la vérification du PIN
- Impact du mode de développement inter-procédural
  - Séparation des contexte d'exécution
  - Attention aux sauts d'appel de fonction

➔Vulnérabilités

- Test d'égalité des octets des PIN utilisateur et PIN carte
- Appels de fonction simples

/03/

Implémentation et Évaluation du Simulateur Embarqué

B – EFS vs. Simulation haut niveau

SAFRAN Morpho

BUT

➔Objectifs

- Etude de deux approches différente d'analyse de code face aux fautes
- Trouver une base commune malgré la différence de niveau d'abstraction
- Comparer les deux approches sur un code commun

➔Intérêts

- Simulation multi-niveau d'une implémentation face aux injection de fautes
- Repérer les faiblesses de chaque approche pour les améliorer
- Quelle approche pour quels besoins ?

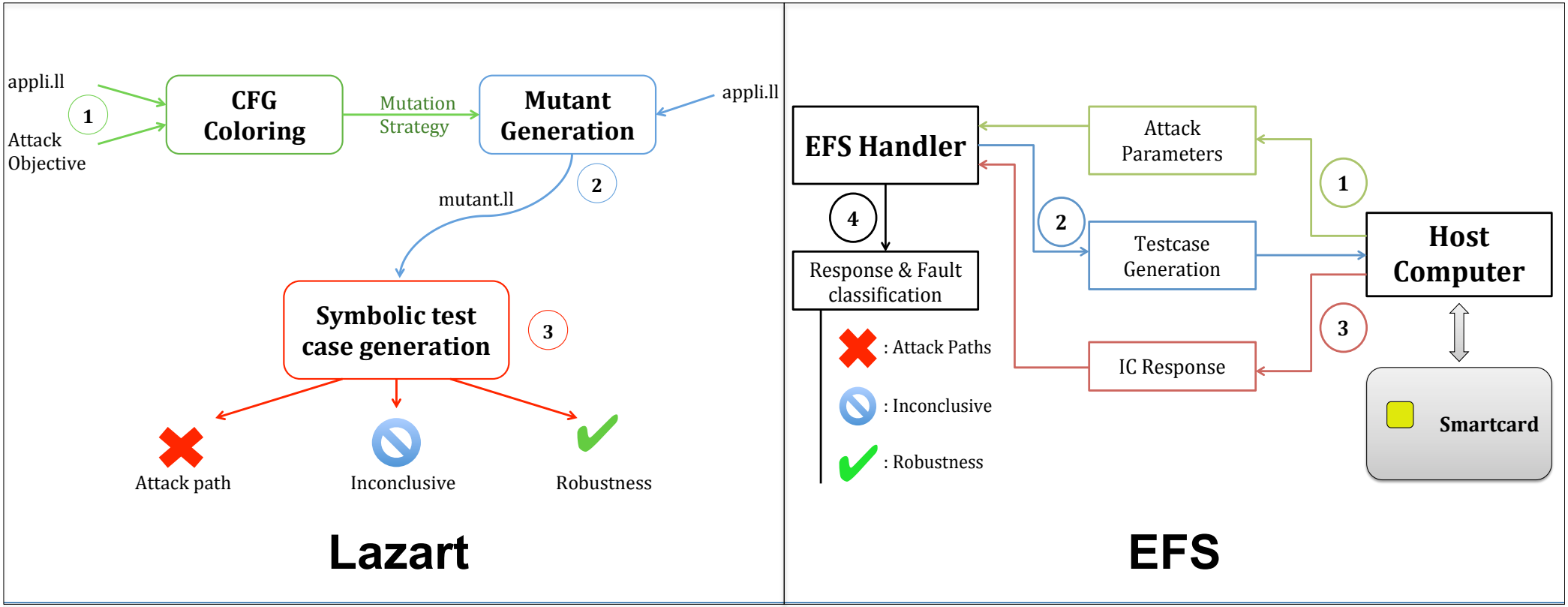
➔Expériences réalisées

- Deux implémentations d'un code de vérification de PIN
- Attaque indépendante avec chaque outil
- Combinaison des deux approches

TRAVAUX COLLABORATIFS - VERIMAG

➔Evaluation de Lazart et de l'EFS

- Lazart : Analyse Statique à haut niveau avec génération de code mutant sur plateforme PC
- EFS : Analyse dynamique à bas niveau sans mutation en système embarqué



PROTOCOLE EXPÉRIMENTAL

➔Code commun

- byteArrayCompare : comparaison de tableau d'octet
- VerifyPIN Local : fonction monolithique
- VerifyPIN Call : fonction inter-procédurale

➔Deux modes de combinaison

- Parallèle – Expériences indépendantes et croisement des résultats
- Séquentielle
  - Premier tour avec Lazart : Identification des zones vulnérables
  - Attaques de ces zones avec l'EFS

➔Intérêt

- Attester de la faisabilité d'une faute d'un niveau d'abstraction à un autre

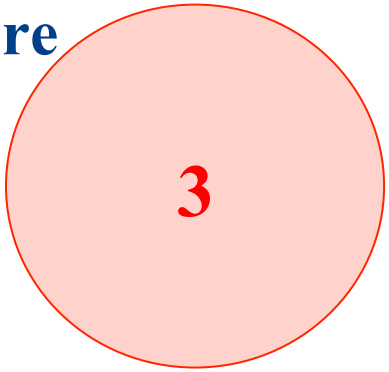
APPROCHE PARALLÈLE ET COUVERTURE

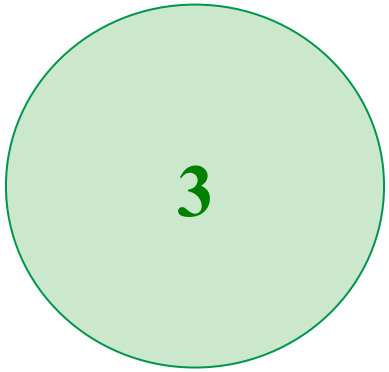
➔Résultats Lazart & EFS

Fault Multiplicity	# of attack	# non-redundant
0	0	0
1	0	0
2	2	2
3	5	0
4	11	1
Total	18	3

# of skipped instructions	# of attacks	Representative attacks
0	0	0
1	1	1
2	2	1
3	1	0
4	4	0
5+	64	1
Total	72	3

➔Couverture





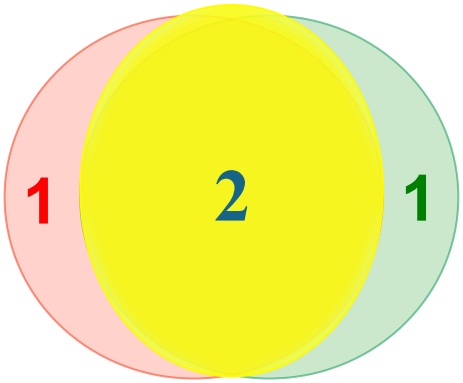
APPROCHE PARALLÈLE ET COUVERTURE

➔Résultats Lazart & EFS

Fault Multiplicity	# of attack	# non-redundant
0	0	0
1	0	0
2	2	2
3	5	0
4	11	1
Total	18	3

# of skipped instructions	# of attacks	Representative attacks
0	0	0
1	1	1
2	2	1
3	1	0
4	4	0
5+	64	1
Total	72	3

➔Couverture

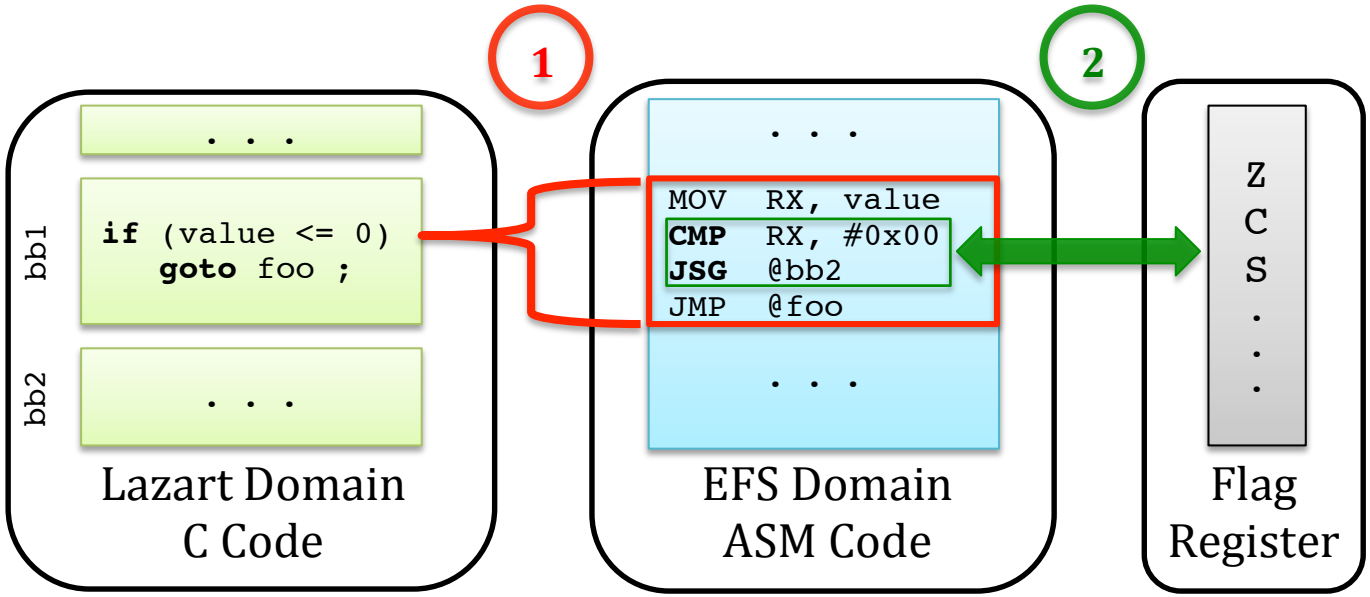


DIVERGENCE ET ABSTRACTION

➔Divergence

- Nombre de test avec Lazart vs EFS → 49 vs 4528 respectivement

➔Différence de niveau d'abstraction



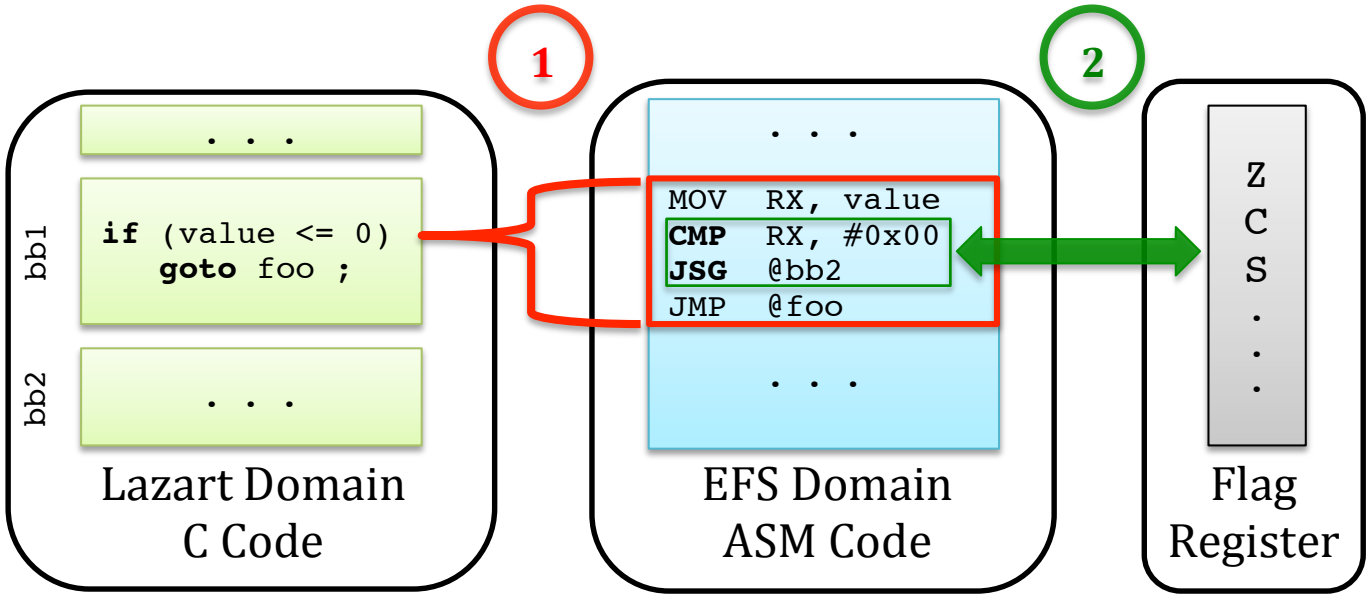
DIVERGENCE ET ABSTRACTION

➔Divergence

- Nombre de test avec Lazart vs EFS → 49 vs 4528 respectivement

➔Différence de niveau d'abstraction

- A une expression en C peut correspondre plusieurs instructions assembleur (1)



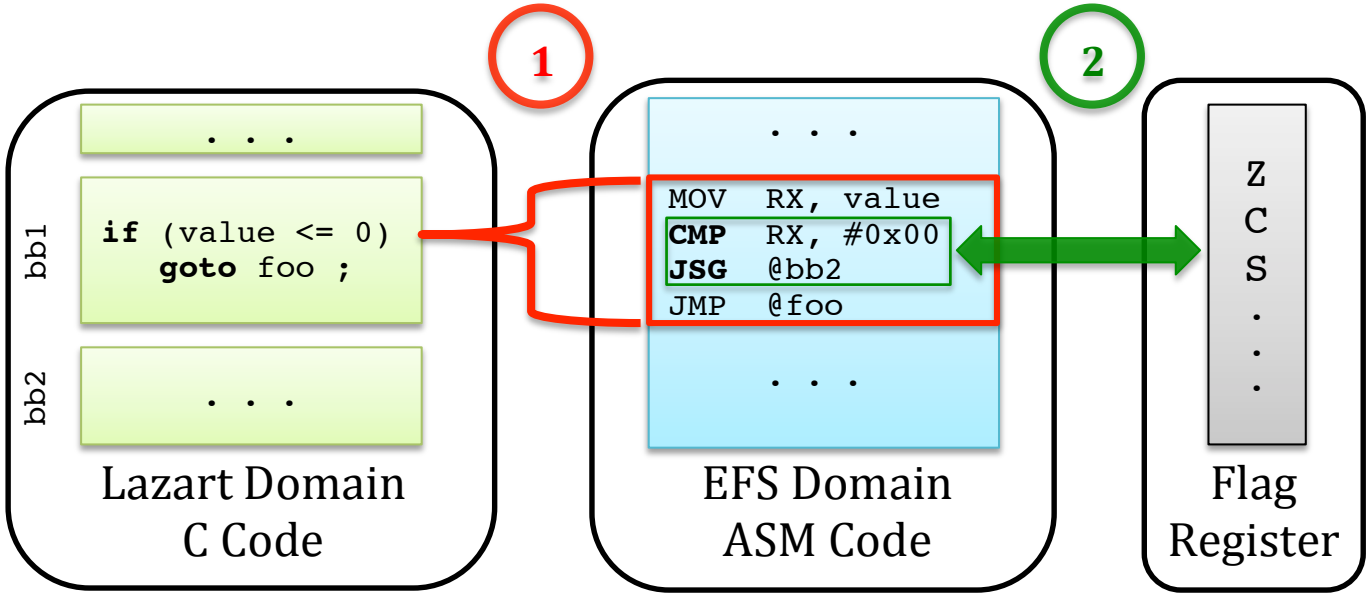
DIVERGENCE ET ABSTRACTION

➔Divergence

- Nombre de test avec Lazart vs EFS → 49 vs 4528 respectivement

➔Différence de niveau d'abstraction

- A une expression en C peut correspondre plusieurs instructions assembleur (1)
- Les opérations très bas niveaux sont totalement abstraites en C (2)



APPROCHE SÉQUENTIELLE - PERFORMANCES

➔ Evaluation de code séquentielle

1. Lazart : Déterminer l'objectif sécuritaire
2. Lazart : Localiser les portions de code sensibles selon la coloration du CFG
3. EFS : cibler les portions de codes reportées (adresses ASM correspondantes)
4. EFS : Attaque des portions de code sur l'espace restreint

➔ Complémentarité de Lazart et de l'EFS sur le VerifyPIN

- Gain d'un facteur 10 en timing pour EFS + Lazart (par rapport à EFS seul)
- Le taux de détection augmente (grâce à l'espace restreint reporté par Lazart)

Approach	# of tests	# of attacks	Timing performance
Lazart	49	18 (3)	< 3s
EFS	4528	72 (2)	$\sim$ 17mn
Lazart + EFS	49 + 720	14 (3)	$\sim$ 1mn30s

*On se limite au saut de 4 instructions avec l'EFS dans cette expérience

/04/

Conclusion et Perspectives

TRAVAUX EFFECTUÉS

➔Implémentation

- Code générique d'appel et de configuration du simulateur embarqué (C)
- Mécanisme de faute à bas niveau (ASM)
- Intégration à des produits existants dans leur phase de test
- Fonctions supplémentaires : trigger side-channel paramétrable

➔Evaluation sur produit réel et microcontrôleur

- Impacts fonctionnels
- Impacts side-channel
- Analyse comportementale en fonction du modèle de faute
- Attaque multiple expérimentale (1 faute simulée + 1 injection laser)
- Perturbation de fonction cryptographiques et sécuritaires (DES, AES)

TRAVAUX PUBLIÉS

➔ Combining High-Level and Low-Level Approaches to Evaluate Software Implementations Robustness Against Multiple Fault Injection Attacks

- *L. Rivière, M.-L. Potet, T.-H Le, J. Bringer, H. Chabanne et M. Puys*
- Foundation on Practice of Security – FPS, Novembre 2014

➔ High-Level Simulation for Multiple Fault Injection Evaluation

- *M. Puys, L. Rivière, J. Bringer et T.-H. Le*
- Quantitative Aspects in Security Assurance – QASA, Septembre 2014

➔ Idea: Embedded Fault Injection Simulator on Smartcard

- *M. Berthier, J. Bringer, H. Chabanne, T.-H. Le, L. Rivière et V. Servant*
- Engineering Secure Software and Systems – ESSoS, Février 2014

AVANCEMENT ACTUEL

➔Travaux en cours…

- Génération et analyse des chemins d'attaques
- Modèles de fautes pratique et intégration pour la simulation
- Interprétation multi-niveaux et mesure de criticité

➔Perspectives pour le simulateur de faute embarqué

- Prise en charge des fautes multiples
- Attaque exhaustive de transactions complètes

FIN

Merci, à vos questions !

lionel.riviere@morpho.com

Ces travaux sont issus de collaborations avec Maël Berthier[1], Julien Bringer[1], Hervé Chabanne[1,2], Thanh-Ha Le[1], Marie-Laure Potet[3], Maxime Puys[1] et Victor Servant[1].

Ces travaux ont été partiellement financés par le Projet ANR Français E-MATA HARI.

[1] SAFRAN Morpho – [2] Télécom ParisTech – [3] Verimag