RÉPUBLIQUE FRANÇAISE
*Liberté*
*Égalité*
*Fraternité*

Inria

cea

université de BORDEAUX

# Analyse automatisée d'une bibliothèque cryptographique

## Détection de failles par canal auxiliaire par analyse statique et symbolique

Duzés Florian

**Master Cryptologie et Sécurité Informatique**

29 août 2025

# 01
# Introduction

# Introduction

## HACL*

*"**H**igh **A**ssurance **C**ryptography **L**ibrary"*[Zin+17] [a] est une bibliothèque cryptographique, écrite en F* ("F star"), implémentant tous les algorithmes de cryptographie modernes et est prouvée mathématiquement sûre.

HACL* est notamment utilisé dans plusieurs systèmes de production tels que Mozilla Firefox, le noyau Linux, le VPN WireGuard...

a. `https://hacl-star.github.io/`

# Introduction - 2

**1996 : Paul C. Kocher, *Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems***

Une mesure précise du temps requis par des opérations sur les clés secrètes permettrait à un attaquant de casser le cryptosystème.

2003 : Brumley et Boneh *Remote Timing Attacks Are Practical*

# Introduction - 2

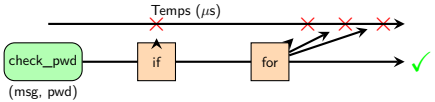
**1996 : Paul C. Kocher, *Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems***

Une mesure précise du temps requis par des opérations sur les clés secrètes permettrait à un attaquant de casser le cryptosystème.

2003 : Brumley et Boneh *Remote Timing Attacks Are Practical*
2011 : Brumley et Tuveri *Remote Timing Attacks are Still Practical*

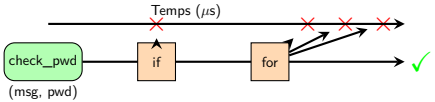
# Petit exemple

```
1   bool check_pwd(msg, pwd){
2   if (msg.length != pwd.length){
3       return False
4   }
5   for(int i = 0; i < msg.length; i++){
6       if(msg[i] != pwd[i]){
7           return False
8       }
9   }
10  return True
11  }
```



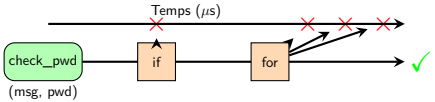
# Petit exemple

```
1   bool check_pwd(msg, pwd){
2   if (msg.length != pwd.length){
3       return False
4   }
5   for(int i = 0; i < msg.length; i++){
6       if(msg[i] != pwd[i]){
7           return False
8       }
9   }
10  return True
11  }
```



# Petit exemple

```
1   bool check_pwd(msg, pwd){
2   if (msg.length != pwd.length){
3       return False
4   }
5   for(int i = 0; i < msg.length; i++){
6       if(msg[i] != pwd[i]){
7           return False
8       }
9   }
10  return True
11  }
```



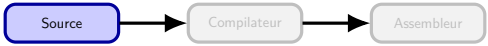

**Opérations influantes :**

- Accès mémoire
- Décalage/rotation de valeurs

- Saut conditionnel
- Division/multiplication

# Structure de travail

# Travail à la source

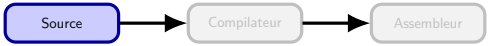



**Programmation en temps constant**

+ Position haut niveau
+ Couverture d'architectures importantes

- Rigueur et conception particulière des actions
- Identification des points de fuites

# Travail à la source




**Programmation en temps constant**

+ Position haut niveau
+ Couverture d'architectures importantes

- Rigueur et conception particulière des actions
- Identification des points de fuites

2024 : Schneider et al., *Breaking Bad : How Compilers Break Constant-Time Implementations*

# Travail avec le compilateur


**Utilisation des compilateurs**

- Constantine - 2021
- Jasmin - 2017
- Raccoon - 2015
- CompCert - 2008 (2019)

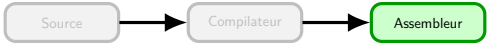
# Travail avec le compilateur


**Utilisation des compilateurs**

- Constantine - 2021
- Jasmin - 2017
- ~~Raccoon~~ - 2015
- CompCert - 2008 (2019)

- Couverture des architectures supportée
- Informations à transmettre
- Spécifications ne sont plus respectées

# Travail en assembleur




**Écrire en assembleur**

+ Efficace
+ Contrôle total

- Spécifique à chaque processeur qui exécute
- Beaucoup de connaissance spécifique au processeur ciblé
- Long à mettre en place - Portabilité faible

# Réalisation

# Réalisation


**Vérification de binaire**

. Approche simple . Approche systématique . Approche automatique

# Réalisation


**Vérification de binaire**

. Approche simple . Approche systématique . Approche automatique

**Érysichthon**

Premier outil ajouté à une intégration continue qui vérifie formellement et complètement une bibliothèque cryptographique.

Analyse HACL* :

| | Prouvé | Attendu |
|---|---|---|
| (%) Fontions sécurisées | 67,96% | 95% |
| (%) Fontions attaquables | 3,08% | 5% |
| (%) Analyse interrompue | 28,96% | 0% |