CHAPITRE 1

Implémentations pour un usage industriel

conception, modèle, difficulté problèmes, specificité des lib...
- motivations - liste des taches a realiser - prise en main - corrections BINSEC - difficultes

1.1 Identification des besoins et spécificités

On a pus voir grâce aux chapitres précédents que la conception et l'implémentation d'une système sécurisé est un problème difficile. Une première étape est de concevoir des primitives et protocoles mathématiquement sécurisés. Une seconde étape est de s'assurer que leurs implémentations sont effectivement sécurisées, d'abord d'un point de vue mathématique contre des attaques logiques (aspect fonctionnel : le code implémente correctement les bons concepts cryptographiques), mais aussi contre des attaques très bas niveau, les attaques temporelles.

Avec l'objectif de concevoir un sytème sûr, il nous faut donc identifier toutes les tâches à réaliser pour arriver à bout de ce projet. En plus de ce travail de planification, l'identification et l'intégration d'outils déjà implémenté nous permettra de d'avancer plus rapidement vers cet objectif.

Point de départ

En reprenant ces deux étapes, on va identifier quels sont nos leviers et nos possibilités pour un développeur pour avancer dans la conception de notre graal.

La première étape de conception de primitives cryptologiques et de protocole n'est pas du ressort du développeur. Elle appartient aux cryptologues et aux chercheurs en sécurité mathématique. Ce sont eux qui concoivent et maintiennent des librairies cryptographiques, des boîtes à outils qui propose les briques de sécurité nécessaires aux systèmes sécurisés.

Plusieurs librairies existent [AHa98 ; Por16 ; Pol+20] et remplissent différents objectifs : rétro-compatiblité, politique temps constant*etc* Notre choix est à réaliser en fonction des spécifités du produits que l'on cherche à déployer.

La seconde étape est à distinguer en deux parties. Cette opération de vérification de la sécurité de l'implémentation peut-être réaliser sur le produit fini et sur les librairies employés par le produit. Comme introduit, cette étape à pour objectif la vérification formelle du code du programme et la vérification matérielle au niveau assembleur.

Utiliser la bibliothèque **Hacl*** [Pol+20 ; Zin+17] permet d'avancer la première étape et la première partie de la seconde étape. Cette bibliothèque a été conçue formellement et vient avec les preuves mathématique de la sécurité de son implémentation. Comme présenté en **??**, cette librairie est programmée en F*. Le projet permet une exploitation en C et en assembleur [Zin+17].

En revanche, la seconde étape de la seconde partie nous demande une vérification au niveau de l'assembleur. Si certaine partie de cette librairie sont codées en assembleur, la majorité du projet reste du F* traduit vers C. Il faut réaliser une analyse. Dans le cadre de cette étude, l'outil d'analyse binaire retenu pour réaliser cette tâche est **Binsec**. Cette outil est implémenté en Ocaml et est maintenu par une équipe de chercheurs ingénieurs géographiquement proche de l'équipe PROSECCO Inria. Cet avantage permet des échanges plus directs et donc une facilité quand à la mise en place du projet.

L'objectif est donc d'analyser Hacl* dans son entièreté. Avec cette analyse complète, si elle est correcte, alors les deux étapes de réalisation d'un système sûr seront réalisées. Cela signifie que la première librairie cryptographique formellement sûre et résistante aux attaques temporelles sera conçus.

Opérations à réaliser

Sans reprendre les explications du fonctionnement de Binsec, voir "ref vers fonctionement de Binsec", l'analyse se réalise sur un fichier binaire à l'aide d'un carnet d'instructions à préciser. Avec ce point de départ, on peut commencer à construire notre carnet de spécifications.

Fichier binaire. Il faut donc des fichiers binaires à fournir à Binsec. Or comme chacun le sait, plus un binaire est imposant, plus son analyse est difficile. Et comme Binsec emploi l'analyse symbolique, explorer un binaire imposant a un coût de mémoire quadratique sur le parcours des instructions du binaire. L'idéal est donc d'analyser plein de petits fichiers binaires.

Analyse complète. Chaque fonction de Hacl* doit être analysée. En poursuivant la condition précédente, on peut essayer de concevoir un binaire par fonction analysé. On distribue ainsi l'analyse et on parcourt ainsi toute les fonctions présentent dans la librairie.

Analyse correcte. Si on se rappel comment fonctionne les optimisations (voir le tableau **??**) il faut faire attention avec certaines optimisations qui simplifient le code par soustraction d'oprations. Le fichier ne doit pas seulement contenir un appel de fonction, il faut une légère mise contexte.

```c
#include <stdlib.h>

#include "Hacl_AEAD_Chacha20Poly1305_Simd128.h"

#define BUF_SIZE 16384
#define KEY_SIZE 32
#define NONCE_SIZE 12
#define AAD_SIZE 12
#define TAG_SIZE 16

uint8_t plain[BUF_SIZE];
uint8_t cipher[BUF_SIZE];
uint8_t aead_key[KEY_SIZE];
uint8_t aead_nonce[NONCE_SIZE];
uint8_t aead_aad[AAD_SIZE];
uint8_t tag[16];

int main (int argc, char *argv[])
{
Hacl_AEAD_Chacha20Poly1305_Simd128_encrypt
    (cipher, tag, plain, BUF_SIZE, aead_aad, AAD_SIZE, aead_key, aead_nonce);
exit(0);
}
```

Code 1 – Code d'anlayse de la fonction Hacl_AEAD_Chacha20Poly1305_Simd128_encrypt, testé lors de la prise main de Binsec et Hacl*

De même, comme nos fichiers analysés vont faire appel à Hacl*, l'emploi de l'option `-static` est nécessaire pour prévenir la mise place de lien vers la librairie partagée. Cette

option ne nuit pas à la qualité de l'analyse, elle permet d'avoir tous les éléments sous la main lorsque l'on déassemble un fichier binaire. Retirer cette option lors de la compilation, c'est entraîner des lourdeurs et rallonger la temps requis pour la vérification manuelle d'un fichier.

Couverture d'architectures. x86_64 et ARM sont les architectures matérielles les plus répandues dans le monde. Étendre l'analyse vers différentes plateformes et observer les différences qui émergent nous permettraient d'avancer dans la direction de la conception d'une librairie cryptographique universelle. On peut alors étendre l'analyse vers d'autres architecture comme PowerPC ou RiscV.

CHAPITRE 2

Érysichton à jamais affamé

- point histoire
- structure / schemas
- usage
- Andrihminir

usage de l'outil, comment ça rend