MRI 2007-2008 – COURS DE CONCEPTION DE SOC

Résumé de l'article « A Domain Specific Language for Cryptography » de Giovanni Agosata et Gerardo Pelosi

Guillaume Libersat

17 décembre 2007

1 Introduction

De plus en plus d'applications cryptographiques sont destinées à être implantées sur des architectures matérielles ou logicielles spécifiques. Ainsi, ces développements se révèlent être des processus qui impliquent de nombreux experts dans des domaines transverses.

Afin d'inciter le développement coopératif et donc de supprimer les erreurs liées à la compréhension tout au long du développement (souvent dûes à l'utilisation de langages génériques comme le *C*), un *DSL*[1] est proposé.

2 Un DSL basé sur Python

Le *DSL* proposé est basé sur le langage Python et peut être utilisé à travers de nombreux niveaux d'abstractions. La philosophie de ce premier est de reposer au maximum sur les concepts de son langage père et de n'étendre que ce qui est nécessaire.

Il apporte cependant de nombreuses améliorations comparé aux langages génériques : il introduit (i) des notations propres au domaine de la cryptographie (Théorie des nombres, Algèbre), (ii) des opérateurs permettant d'indiquer les opérations optimisables (qui sont normalement difficiles à détecter), (iii) des types et structures de données spécifiques au domaine.

2.1 Les types

La plupart des algorithmes basent leur sécurité sur la dureté de problèmes mathématiques, c'est pourquoi les concepteurs sont souvent amenés à manipuler des structures propres à l'algèbre (anneaux, ...).

Pour la **représentation** de ces dernières, ce *DSL* propose comme réponse un système malléable permettant de décrire facilement des types (on pourra donc associer à chaque type une précision fixée ou illimitée). Par exemple, on pourra décrire un entier 32 bits par "int.32". On pourra aussi aisément définir des tableaux ou effectuer des conversions sans les risques associés au transtypage.

La langage ajoute aussi l'opérateur "mod" qui permet d'utiliser l'**arithmétique modulaire** en forçant des conversions entières.

Enfin, ce *DSL* propose une solution pour exprimer des **polynômes de degrés arbitraires**. Ainsi, l'opérateur "polynomial" permet de spécifier qu'un tableau est en fait un polynôme (ex : "polynomial mod 5 int [3]")

[1] DSL pour Domain Specific Language

2.2 Les opérateurs

En ce qui concerne les **opérateurs scalaires**, le *DSL* inclue tous les éléments classiques et vient ajouter une opération de substitution (“S-box”) via l’opérateur “$”.

Du côté des **opérateurs vectoriels**, des enrichissements sont aussi proposés. Tout d’abord l’opérateur “\” permet de dupliquer les éléments d’un tableau sans l’aplatir ([1,2] donne [1,2,1,2]). Ensuite, des notions comme la rotation sont introduites.

On notera que des précautions ont été prises afin de (i) permettre l’utilisation des opérateurs scalaires sur les vecteurs et (ii) faciliter la manipulation d’éléments de tailles différentes.

2.3 Les fonctions

Afin de prendre en compte la cible d’implantation de l’algorithme, ce *DSL* ajoute la possibilité de typer les paramètres d’une fonction (ce qui n’est pas le cas dans Python). Ainsi, il est par exemple possible de savoir si une donnée peut être rangée dans un registre.

La langage propose aussi des fonctions utilitaires comme “perm” (permettant la permutation de tableaux) qui sont généralement exhaustives à programmer. Enfin, il met à disposition des primitives pour la génération de **nombres aléatoires typés**.

3 Analyse

3.1 Points forts

Le *DSL* proposé a pris le pari de se baser sur un langage éprouvé, reconnu et déjà maîtrisé par un grand nombre de programmeurs. De plus, Python est simple à apprendre et peut être lu facilement. Ce dernier dispose aussi d’une bonne intégration avec SystemC (SystemPy), ce qui accélère encore la transformation des prototypes vers un code cible.

Ce *DSL* permet donc de proposer un ensemble facile à maîtriser permettant de produire rapidement des algorithmes (langage expressif et concis) tout en assurant la transmission de connaissance tout au long du processus de développement.

3.2 Originalité et positionnement

Il existe très peu de *DSL* pour la cryptographie à l’heure actuelle. *Cryptol*, bien qu’il soit complet, permet très difficilement d’effectuer du co-design (syntaxe très inhabituelle et spécifique, nécessitant un apprentissage conséquent de la part de tous les participants d’un projet)

CAO est une autre proposition qui se base sur le *C* et *Occam*. Le *DSL* proposé offre une expressivité de plus haut-niveau ainsi que des concepts inexistants en *C* (meta-classes, etc).

D’un point de vue des productions, ce *DSL* produit des algorithmes beaucoup plus courts en terme de nombre de lignes sans pour autant obfusquer le code.

3.3 Critique personnelle

Personnellement, j’ai trouvé que l’approche permet effectivement pour un non spécialiste de comprendre l’algorithme beaucoup plus facilement (clarté, langage “habituel”, concision) sans pour autant imposer un langage complexe pour les mathématiciens (il rempli donc son but).

L’implantation courante est cependant minimaliste et n’apporte pas d’outils vraiment exploitables qui pourraient effectuer des vérifications et donc réduire les erreurs de conception.