Architecture et développement d'une librairie unifiée de cryptanalyse

De l'état de l'art à une approche fonctionnelle modulaire

William MATHEY
EPITA - Laboratoire de Recherche de l'EPITA (LRE)
19 juillet 2026

Table des matières

01

Introduction

Contexte : l'évolution de la cryptanalyse

Des approches manuelles aux approches automatisées

- **Ad-hoc** : chiffrements de César, Enigma, attaques manuelles Biryukov et Nikolić, 2010 ; Fouque et al., 2013
- **Modélisation** : SAT, (M)ILP, Programmation par Contraintes (CP) pour AES, etc. Gerault et al., 2016 ; Mouha et al., 2011
- **Automatisation** : devenue nécessaire face à la complexité des chiffrements modernes

Les bibliothèques récentes

- **TAGADA** Delobel et al., 2023 ; Libralesso et al., 2021 : manipulation par mots
- **CLAASP** Bellini et al., 2023 : manipulation par bits
- **OCP** : approche mathématiques
- **BONC** : concervation de contexte

Limitations des approches existantes

Problèmes identifiés

- **TAGADA** :
  - Manipulation par **mots** uniquement → impossible de faire des opérations booléennes fines
  - Graphes illisibles après quelques tours (exemple : 3 tours d'AES)
  - Difficulté à retrouver les états/opérateurs après modélisation.
- **CLAASP** :
  - Manipulation par **bits** → perte du contexte arithmétique (mots, entiers modulaires)
  - Booléanise tout → impossible de faire des additions modulaires
- **BONC** :
  - En développement, encore immature
  - Tente de conserver un contexte global mais pas encore abouti
- **OCP** :
  - Implémentation trop éloignée des problématiques réelles
  - Pas adaptée à la cryptanalyse pratique

Illustration : TAGADA

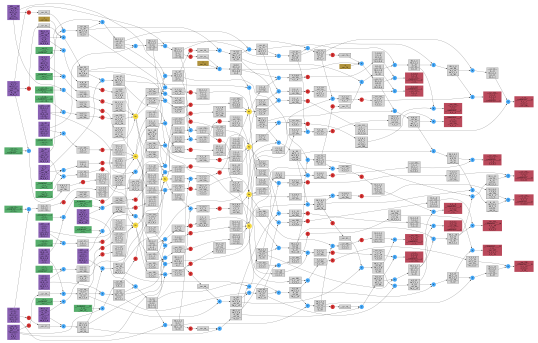


Figure 1 : Schéma de 3 tours d'AES-128 avec TAGADA.

Problème : impossible de retrouver les états/opérateurs après modélisation.

Illustration : Crypanalystes

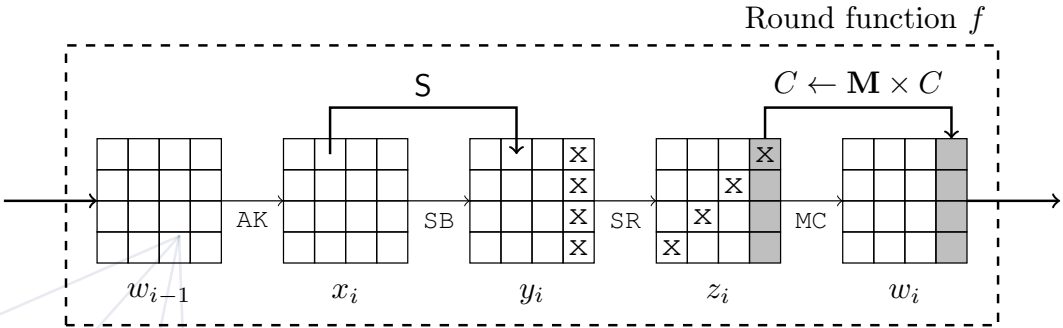


Figure 2 : Schéma d'un tour d'AES-128.

Autres limitations majeures

Manque de modularité et de généricité

- **Impossible d'enchaîner** des tours de chiffrements différents (ex : AES + Midori + SPECK dans la même analyse)
- **Modélisations différentes** : SAT, MILP, CP → code non réutilisable
- **Ajout de couches "invisibles"** : pour aligner des opérations, certaines bibliothèques ajoutent des couches qui n'existent pas dans le chiffrement original
- **Unification difficile** : familles SPN/Feistel/ARX traitées séparément

02

Problématique et Objectifs

Comment créer une bibliothèque unifiée, modulaire et efficace pour la cryptanalyse des chiffrements symétriques et asymétrique ?

Objectif principal

Développer une bibliothèque unifiée de cryptanalyse, modulaire, maintenable et facile à utiliser.

Objectifs

Sous-objectifs concrets

- **Visualisation** : représenter graphiquement n'importe quel chiffrement (ex : 7 tours d'AES suivis de 3 tours de Midori)
- **Conservation du contexte** : mots, bits, arithmétique modulaire simultanément
- **Génération d'aléa** : création automatique d'instances de chiffrement
- **Modularité** : enchaînement facile de primitives différentes
- **Extensibilité** : ajout facile de nouveaux chiffrements
- Offrir une visualisation contrôlée (sélection de tours spécifiques)
- Permettre la génération d'aléa et la composition d'opérateurs ensemblistes.

Planning du semestre

Février - Mars :

- Finalisation de l'état de l'art
- Identification des besoins fonctionnels

Avril :

- Recherche itérative de modèles d'architecture

Mai - Juin :

- Conception de l'architecture retenue
- Définition des interfaces (Domaines, Opérateurs, Variables)

Juillet :

- Intégration des premiers chiffrements (AES, Midori, SPECK)
- Preuve de concept (POC) fonctionnelle en Python
- Visualisation et tests

03

Recherche itérative de modèles

Première approche : code impératif

Pourquoi ?

- Semblait plus naturel (une fonction = une action).
- Facile à écrire pour un chiffrement donné.

Limites

- **Très difficile à maintenir.**
- **Manque de modularité** : impossible de réutiliser des blocs.
- Obligerait de générer du code sur mesure pour chaque nouveau chiffrement.

Seconde approche : orientée objet classique

Principes

- Travailler **sur le chiffré globalement**.
- Découpage uniquement si nécessaire (SBox, ShiftColumn).
- Opérations **simultanées** (ex. SBox sur tous les bytes). (Grand gain de complexité)
- **Builders** : un par chiffrement (DES, AES, Midori, Skinny, SPECK, Wrap).
- **Visiteurs** pour l'exécution.

Limites

- Liaison complexe entre variables et opérateurs.
- Vérification des ensembles de définition (domaines) fastidieuse.
- Ajout d'un nouveau chiffrement = nouveau builder + nouveau visiteur.
- Difficulté à composer des chiffrements trop différents (symétrique + asymetrique par exemple)

04

Solution retenue : approche fonctionnelle

Changement de paradigme : tout est variable

Principe fondamental

- On ne manipule que des **variables**.
- Les **domaines** (ensembles de définition) sont attachés aux **variables**, et on surcharge les opérateurs.
- Chaque appel à une fonction est une **instance différente** (évite les conflits de liaison).

Exemple : y = f(x)

Une fonction $f : \mathbb{G}_{2^n} \times \mathbb{G}_{2^n} \to \mathbb{G}_{2^n}$ est représentée par :

$$x \leftarrow f \longleftrightarrow y$$

où x et y sont des variables, et f est un opérateur.

Opérateurs ensemblistes et domaines

Définition d'une variable

Une variable est définie par :

- Un **domaine** (ensemble de définition).
- Une **entrée** (opération qui l'a produit (peu etre nulle)).
- Une **liste de sorties** (opérations dépendantes (peu etre vide)).

Le domaine comme interface

Le domaine est une **interface** ou **classe abstraite** avec des **implémentations concrètes** :

- $\mathbb{F}_{2^n}$: corps binaire de taille n
- $\mathbb{Z}/p\mathbb{Z}$: entiers modulo p
- ...

Chaque opérateur (`ADD`, `TIMES`, ...) est **surchargé** selon le domaine.

Opérateurs ensemblistes et domaines

Exemple : opérateur ADD

L'opérateur ADD est surchargé selon le domaine de x et y :

- ADD : $\mathbb{F}_2^n \times \mathbb{F}_2^n \to \mathbb{F}_2^n$,
  $(x, y) \mapsto x \oplus y$ (XOR bit à bit)
- ADD : $\mathbb{Z}/p\mathbb{Z} \times \mathbb{Z}/p\mathbb{Z} \to \mathbb{Z}/p\mathbb{Z}$,
  $(x, y) \mapsto x + y \mod p$ (addition modulaire)

Gestion des domaines incompatibles

Si les domaines de x et y sont **différents** :

- On lève une erreur (par défaut)
- Ou on effectue une **conversion implicite** si elle est définie

Avantage de cette approche par domaine

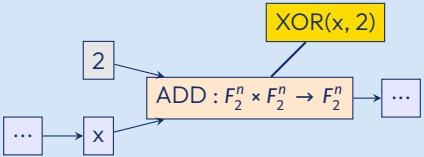
Unification des opérations

- La même syntaxe ADD(x, y) fonctionne **dans tous les contextes**
- On peut **mélanger** des opérations de différents domaines
- On conserve le **contexte** de chaque variable

Optimisation possible par curryfication

Curryfication implicite

Étant donné l'approche choisie, on peut imaginer considérer toute fonction prenant une constante comme paramètre comme une fonction propre :



Pourquoi cette optimisation

Pourquoi faire ça ?

- **Réduction du nombre de paramètres** : simplification des graphes
- **Spécialisation** : on peut pré-calculer certaines opérations
- **Optimisation** : les constantes peuvent être fusionnées

Limitations

- **Perte de généralité** : une fonction curryfiée n'est plus réutilisable avec d'autres constantes

Solution adoptée

- **On ne curryfie donc pas** → On cherche justement de la généralité pour avoir plus de modularité

Avantages de la nouvelle implémentation

Modularité et flexibilité

- **Simplicité** : tout est variable, pas de contextes implicites.
- **Modularité** : on chaîne des graphes de chiffrements quelconques.
- **Compatibilité** : symétrique/asymétrique géré uniformément.
- **Visualisation contrôlée** : on peut sélectionner uniquement certains tours (ex. tours 2 et 3). En se déplaçant dans les liasons entre varialbes.
- **Code maintenable** : ajout facile de nouveaux chiffrements
- **Pas de couches fantômes** : on n'ajoute pas d'opérations inexistantes

Simplicité de cette implémentation

Composants clés

- **Builders** : un par chiffrement + un builder global.
- **Visiteurs** : exécution récursive (gauche ou droite/gauche) pour l'évaluation.
- **Domaine** : interface pour chaque opérateur (ADD, TIMES, ...).
- **Variable** : unité de base, avec son domaine et ses dépendances
- **Opérateur** : fonction qui transforme des variables en variables

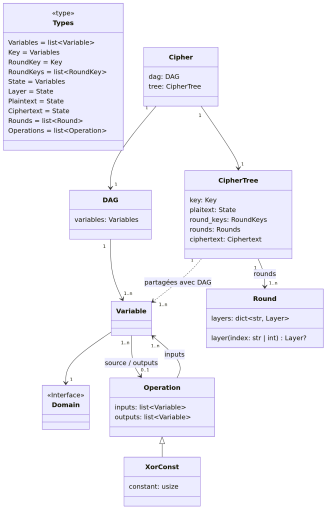


Figure 3 : Diagramme UML simplifié de l'architecture

05

Conclusion

Conclusion

Où nous en sommes

- **Implémentation pas encore terminée** : POC en cours de développement
- Architecture fonctionnelle **validée conceptuellement**
- Reflexion sur la représentation et la génération d'aléa

Pourquoi cette solution va fonctionner

- **Modularité** : chaque composant est indépendant et interchangeable
- **Flexibilité** : on peut représenter n'importe quelle primitive
- **Conservation du contexte** : les domaines préservent l'information
- **Visualisation lisible** : on contrôle ce qu'on affiche
- **Extensibilité** : ajout de nouveaux chiffrements sans refactorisation

Une bibliothèque unifiée, maintenable, et adaptable à toutes les familles de chiffrement.

Des Questions ?

Bibliographie i

- Bellini, E., Gerault, D., Grados, J., Huang, Y. J., Rachidi, M., Tiwari, S., & Makarim, R. H. (2023)
  CLAASP : a Cryptographic Library for the Automated Analysis of Symmetric Primitives.
- Biryukov, A., & Nikolić, I. (2010)
  Automatic search for related-key differential characteristics in byte-oriented block ciphers : Application to AES, Camellia, Khazad and others.
  Annual International Conference on the Theory and Applications of Cryptographic Techniques.
- Delobel, F., Derbez, P., Gontier, A., Rouquette, L., & Solnon, C. (2023)
  A CP-based Automatic Tool for Instantiating Truncated Differential Characteristics .
  LNCS.
- Fouque, P.-A., Jean, J., & Peyrin, T. (2013)
  Structural evaluation of AES and chosen-key distinguisher of 9-round AES-128.
  Annual Cryptology Conference.

Bibliographie ii

- Gerault, D., Minier, M., & Solnon, C. (2016)
  Constraint programming models for chosen key differential cryptanalysis.
  International conference on principles and practice of constraint programming.

- Libralesso, L., Delobel, F., Lafourcade, P., & Solnon, C. (2021)
  Automatic Generation of Declarative Models for Differential Cryptanalysis.
  LIPIcs–Leibniz International Proceedings in Informatics.

- Mouha, N., Wang, Q., Gu, D., & Preneel, B. (2011)
  Differential and linear cryptanalysis using mixed-integer linear programming.
  International Conference on Information Security and Cryptology.