# Identification d'Algorithmes Cryptographiques dans du Code Natif

Pierre Lestringant

Paris, le 31 Mai 2018

Travaux de thèse réalisés à AMOSSYS,
sous la direction de Pierre-Alain Fouque et l'encadrement de Frédéric Guihéry.

# Motivation

Les algorithmes cryptographiques réalisent des fonctions de sécurité.

- évaluer la robustesse de leur mise en pratique.
  *exemple: choix des algorithmes, solidité des implémentations*
- supprimer les protections mises en place.
  *exemple: déchiffrer les communications d'un malware*

Pouvoir **identifier** et **localiser** les algorithmes cryptographiques

# Solutions Existantes

## Solutions Opérationnelles:

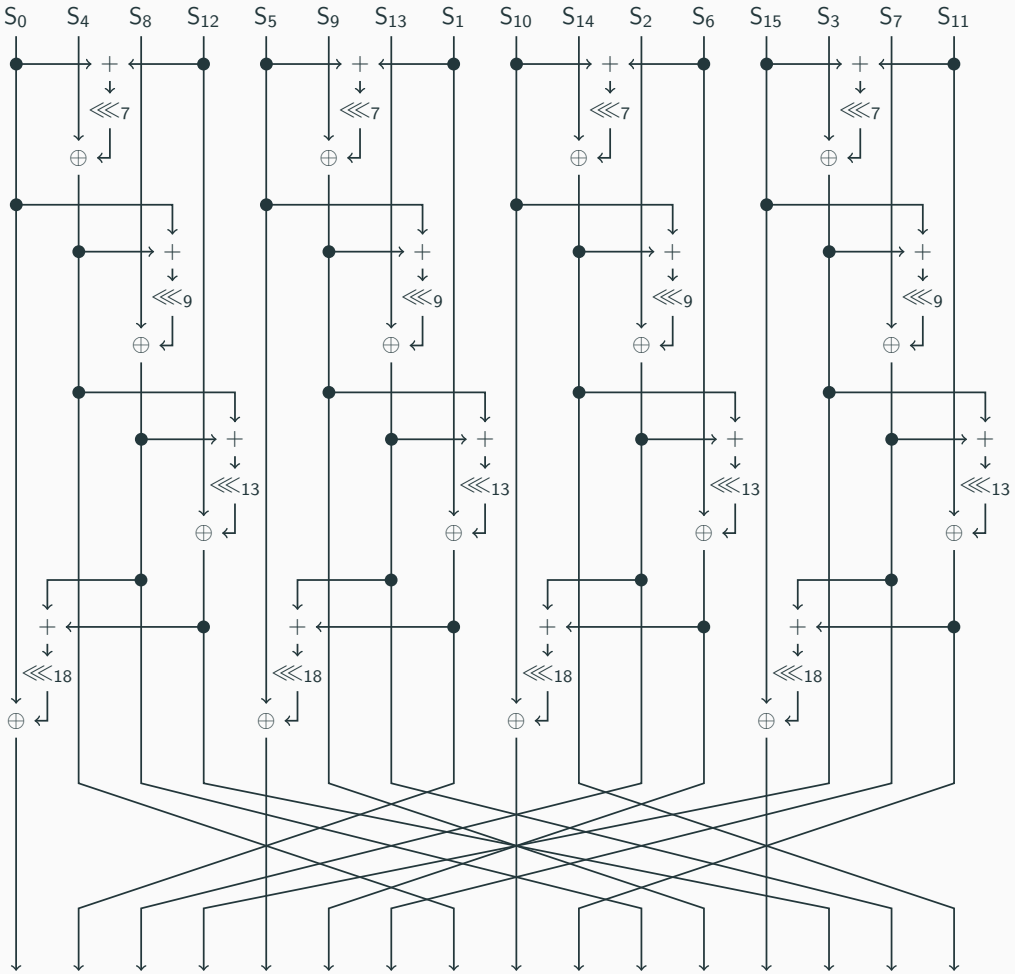
- Heuristiques simples.
  *exemple: constantes et mnémoniques spécifiques, nombre d'exécutions*
- Identification de bibliothèques liées statiquement.
  *exemple: FLIRT, zignature*

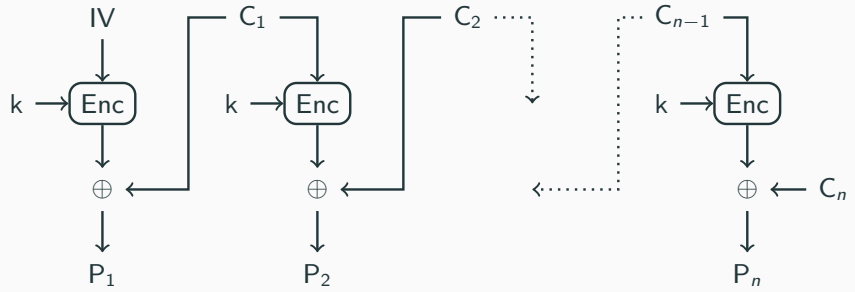
## Solutions Académiques:

- Utilisation de la relation liant les valeurs d'entrée et de sortie.
  Gröbert *et al.* 2011, Zhao *et al.* 2011, Aligot Calvet *et al.* 2012
- Utilisation de la propriété de diffusion.
  TaintScope Wang *et al.* 2010, CipherXRay Li *et al.* 2014

# Méthodes d'Identification

Conception de deux méthodes: une pour les primitives symétriques et une pour les modes opératoires



# Représentation Interne: Graphe de Flot de Données

# Flot de Contrôle

**Observation:** peu de branches conditionnelles dans les algorithmes de cryptographie symétrique.

**Conséquences:**

- le flot de contrôle ne permet pas une identification précise.
- le chemin d'exécution change peu d'une exécution à l'autre.

**Hypothèse de code séquentiel**

**Implémentation:** collecte d'une trace d'exécution (par *exemple* à l'aide d'un framework d'instrumentation dynamique).

Les analyses porteront sur des segments de la trace d'exécution.

# Graphe de Flot de Données

**Idée:** représenter sous forme de graphe les expressions calculées par une suite d'instructions.

**Définition:**

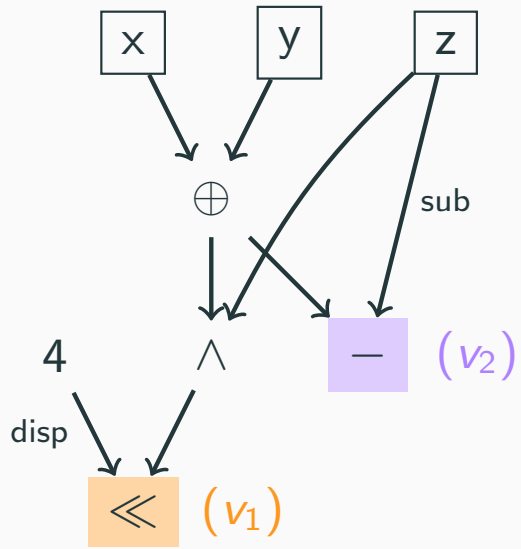
Un DFG $G$ se caractérise par $(V_G, E_G, src_G, dst_G, labV_G, labE_G)$ où:

- $(V_G, E_G, src_G, dst_G)$ est un multigraphe orienté acyclique ;
- $labV_G$ associe à un nœud une opération de $\Sigma$ / une variable de $X$ ;
- $labE_G$ indexe les arêtes partageant une même destination.

A chaque nœud $v$ de $G$ est associé une expression:

$$term_G(v) = \begin{cases} labV_G(v), & \text{si } labV_G(v) \in X \\ labV_G(v)(term_G(v_1), ..., term_G(v_n)), & \text{si } labV_G(v) \in \Sigma \end{cases}$$

# Exemple - Graphe de Flot de Données




$$term_G(v_1) = \ll(\wedge(z, \oplus(x, y)), 4)$$

$$term_G(v_2) = -(\oplus(x, y), z)$$

# Opérations

Un ensemble d'opérations restreint.

| | |
|---|---|
| Arithmétique: | $+$, `divr`, `divq`, `imul`, $\times$, `neg`, $-$ |
| Logique bit à bit: | $\wedge$, $\neg$, $\vee$, $\oplus$ |
| Bit shift: | $\circlearrowleft$, $\circlearrowright$, $\ll$, `shld`, $\gg$, `shrd` |
| Modification taille: | `movzx`, `part18`, `part28`, `part116` |
| Mémoire: | `load`, `store` |

# Nœud Terminaux - Évaluation

**Définitions:**

Les nœuds affectés à un registre à la fin du segment et ceux portant le label `store`, sont appelés nœuds terminaux.

*(ensemble noté $Root_G$)*

Une évaluation $\theta$ est une fonction de l'ensemble des variables $X$ vers un ensemble de valeurs $D$.

Par récurrence on étend $\theta$ à tous les nœuds de $G$: *(noté $\theta_G$)*

$$\theta_G(v) = \begin{cases} \theta(labV_G(v)), & \text{si } labV_G(v) \in X \\ labV_G(v)(\theta_G(v_1), ..., \theta_G(v_n)), & \text{si } labV_G(v) \in \Sigma \end{cases}$$

# Conclusion - DFG

**Intérêts** de cette représentation:

- réécriture d'expressions *(normalisation)*
- recherche d'expressions *(signatures, slicing)*
- visualisation d'expressions *(interprétation des slices)*

Représentation utilisée pour la compilation pouvant être formalisée sous forme de *term graphs* ou de *jungles*.

# Identification de Primitives

# Principe - Identification Primitive

**Objectifs:**

- précision *(localisation des paramètres, du flot de données)*
- robustesse *(différentes compilations, implémentations)*

---

**Recherche d'expressions**

- Étape 1: **Normalisation**
  > Objectif: réduire le nombre d'expressions *équivalentes* à identifier.
  > Solution: réécriture du DFG.
- Étape 2: **Recherche de Signatures**

# Formalisation

**Définition:** *(formalisation de la notion d'expressions équivalentes)*

Deux DFGs $G$ et $H$ sont similaires de façon observable si:

- $labV_G(V_G) \cap X = labV_H(V_H) \cap X$
- $\theta_G(Root_G) = \theta_H(Root_H)$ pour toute évaluation $\theta$

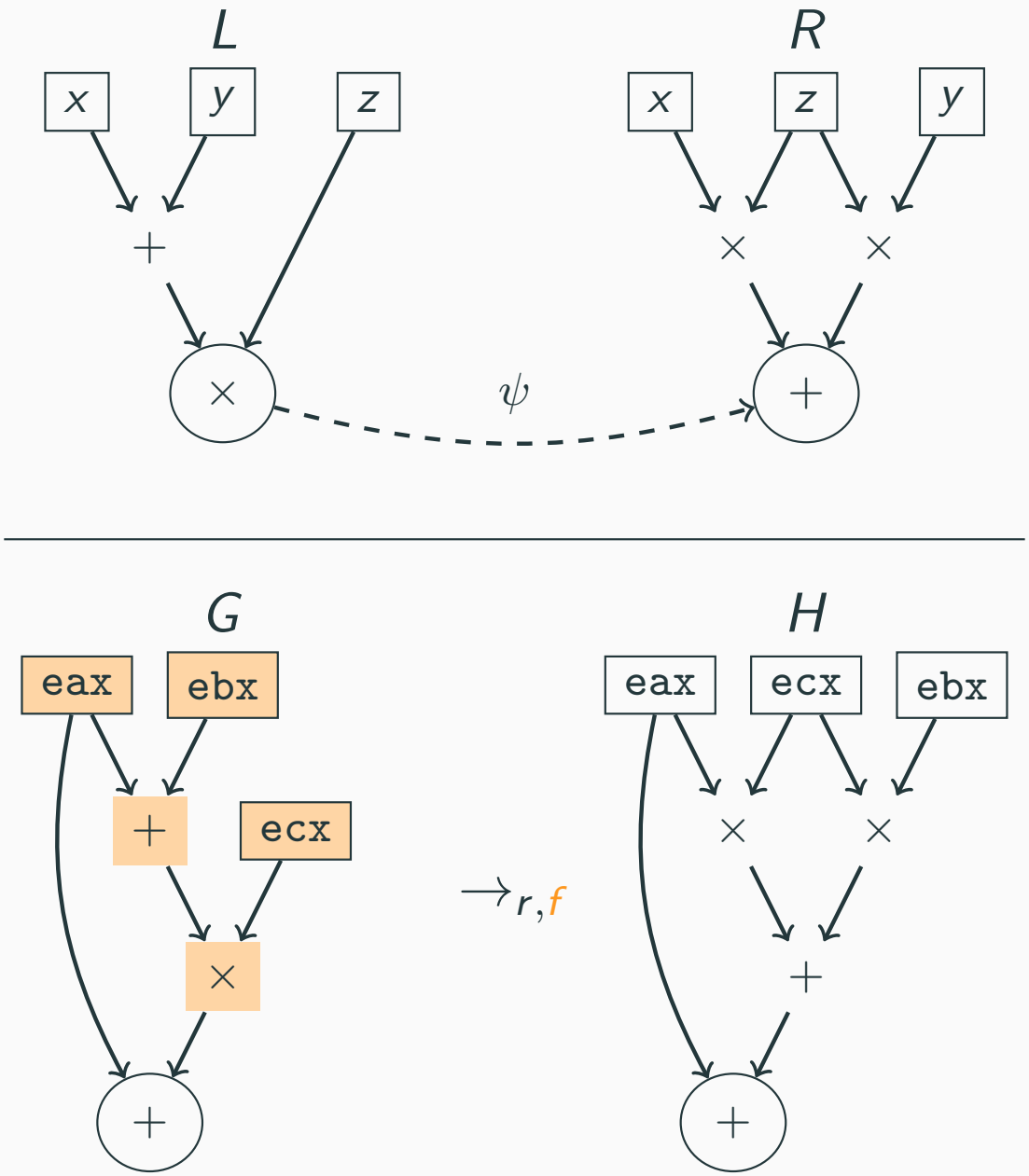
# Formalisation

**Définition:**

Soient deux DFGs $G$ et $H$, un morphisme de $G$ vers $H$ se définit par deux fonctions: $f_V : V_G \to V_H$ et $f_E : E_G \to E_H$ telles que:

- $f_V \circ src_G = src_H \circ f_E$ et $f_V \circ dst_G = dst_H \circ f_E$
- $labV_G = labV_H \circ f_V$ et $labE_G = labE_H \circ f_E$

# Exemple - Étape de Réécriture



# Système de Réécriture

Système de Réécriture: $\rightarrow = \rightarrow_{r1} \cup \rightarrow_{r2} \cup \rightarrow_{r3}$

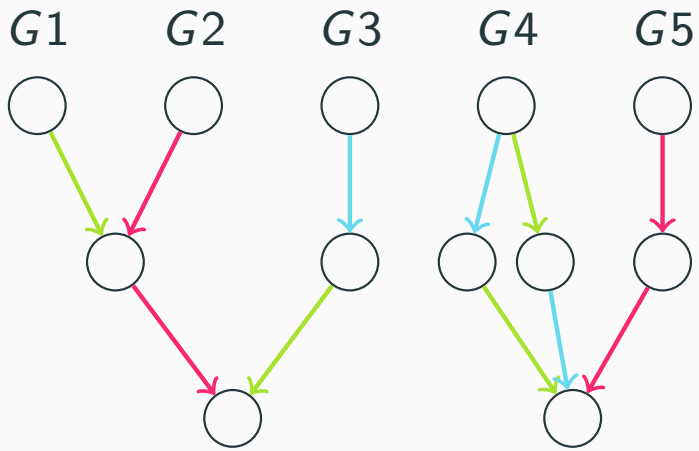

- : règle 1 $\rightarrow_{r1}$
- : règle 2 $\rightarrow_{r2}$
- : règle 3 $\rightarrow_{r3}$

# Système de Réécriture

Système de Réécriture:

$\rightarrow = \rightarrow_{r1} \cup \rightarrow_{r2} \cup \rightarrow_{r3}$

- : règle 1 $\rightarrow_{r1}$
- : règle 2 $\rightarrow_{r2}$
- : règle 3 $\rightarrow_{r3}$




**Définition**:

$\bar{G}$ est une forme normale de $G$ si:

- $G \rightarrow^* \bar{G}$
- il existe aucun élément $H$ tel que $\bar{G} \rightarrow H$

# Système de Réécriture - Terminaison

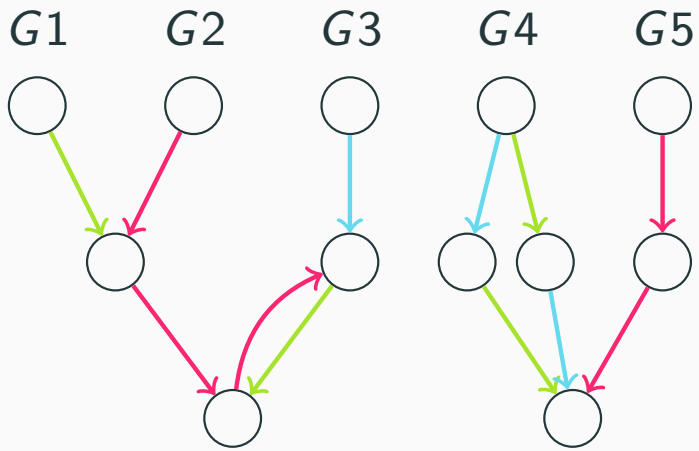
Système de Réécriture:

$\rightarrow = \rightarrow_{r1} \cup \rightarrow_{r2} \cup \rightarrow_{r3}$

■ : règle 1 $\rightarrow_{r1}$

■ : règle 2 $\rightarrow_{r2}$

■ : règle 3 $\rightarrow_{r3}$




**Définition**:

La relation $\rightarrow$ termine s'il n'existe aucune suite infinie $(G_n)$ telle que:

$$G_0 \rightarrow G_1 \rightarrow G_2 \rightarrow \dots$$

# Système de Réécriture - Convergence

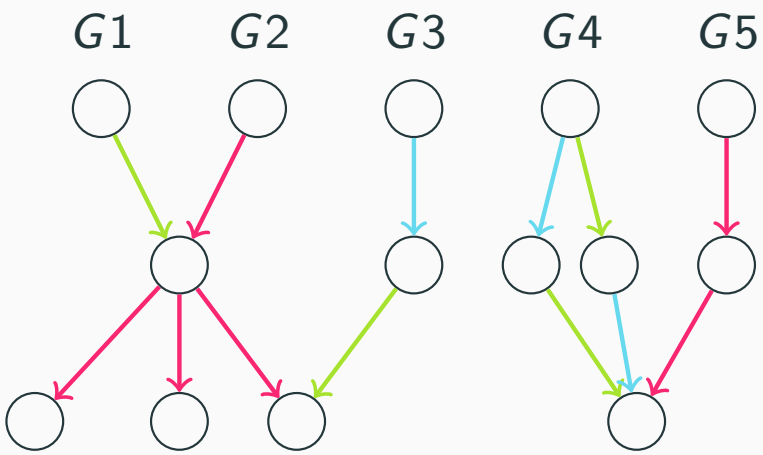
Système de Réécriture:

$\rightarrow = \rightarrow_{r1} \cup \rightarrow_{r2} \cup \rightarrow_{r3}$

■ : règle 1 $\rightarrow_{r1}$

■ : règle 2 $\rightarrow_{r2}$

■ : règle 3 $\rightarrow_{r3}$




**Définition**:

La relation $\rightarrow$ est convergente si chaque élément possède une unique forme normale.

# Système de Réécriture - Préservation Similarité Observable

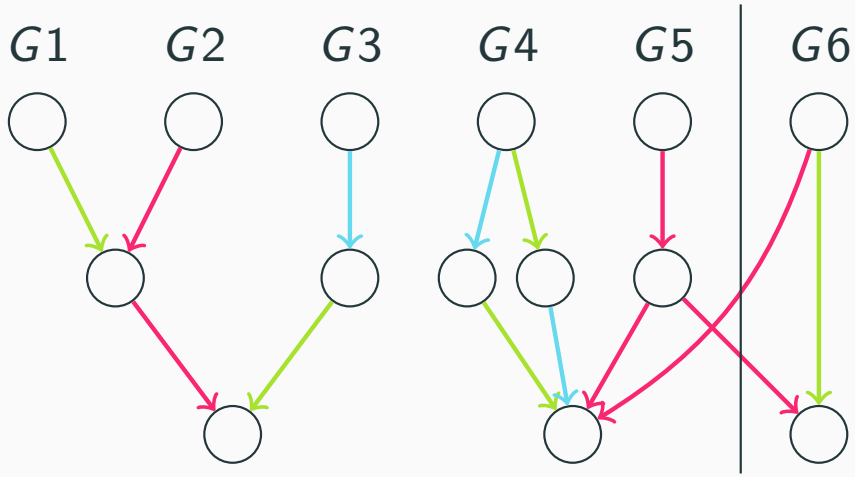
Système de Réécriture:

$\rightarrow = \rightarrow_{r1} \cup \rightarrow_{r2} \cup \rightarrow_{r3}$

: règle 1 $\rightarrow_{r1}$

: règle 2 $\rightarrow_{r2}$

: règle 3 $\rightarrow_{r3}$




**Définition**:

La relation $\rightarrow$ préserve la similarité observable si pour tout élément $G$, $\bar{G}$ est similaire de façon observable à $G$.

**Propriété:**

Soit une étape de réécriture de $G$ vers $H$ suivant $r = (L, R, \psi)$,
*si* $\theta_{L|Root_L} = \theta_R \circ \psi$ pour toute évaluation $\theta$,
*alors* $G$ et $H$ sont similaires de façon observable.

# Système de Réécriture - Sensibilité au Contexte

Il existe un morphisme injectif de $S$ vers $\bar{G}$ **ssi** il existe un DFG $H$ similaire de façon observable à $S$ et un morphisme injectif de $H$ vers $G$.

**Hypothèse:** tous les DFGs d'une classe d'équivalence convergent vers une même forme normale $S$. *(normalisation idéale)*

**Contrainte:** le segment d'analyse contient le contexte d'appel.

**Sensibilité au contexte:** la normalisation ne préserve pas les morphismes $(f : G \to H \not\Rightarrow \exists f' : \bar{G} \to \bar{H})$.

# Mécanismes de Normalisation

Un même ensemble de règles pour toutes les primitives.

1. Constant Folding
2. Détection d'Expressions Constantes
3. Divers Règles
4. Suppressions des Sous-Expressions Communes
5. Simplification des Accès Mémoire
6. Distribution des Opérations avec Opérandes Constantes
7. Elargissement d'Expressions
8. Memory Coalescing
9. Simplification des Expressions Affines
10. Fusion des Opérations avec Opérandes Constantes
11. Normalisation des Opération Commutatives & Associatives

# Principe - Identification Primitives

**Objectifs:**

- précision *(localisation des paramètres, du flot de données)*
- robustesse *(différentes compilations, implémentations)*

---

**Recherche d'expressions**

- Étape 1: **Normalisation**
- Étape 2: **Recherche de Signatures**

  Objectif: identifier efficacement des expressions connues.
  Solution: recherche de morphismes et signatures composées.

# Signature Simple

**Définition:**

Une signature est un DFG $S$ complété par:

- un symbole $\omega_S$ ;
- une suite de nœuds $I_S$ représentant les paramètres d'entrée ;
- une suite de nœuds $O_S$ représentant les paramètres de sortie.

Énumération des morphismes injectifs grâce, par exemple, à l'algorithme d'Ullmann.

---

**Que faire si plusieurs formes normales?** *Exemple* de la rotation:

$\circlearrowleft(x, c)$

$\vee(\ll(x, c), \gg(x, 32 - c))$

# Signature Composée

Identifier les différentes parties d'une expression séparément.

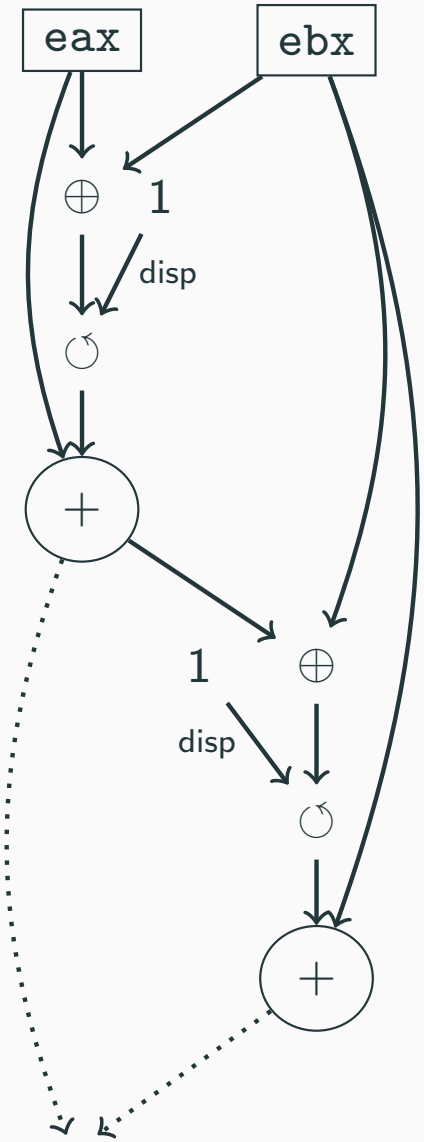
**Définition:**

Une signature composée contient un ou des nœuds dont le label est un symbole de signature.

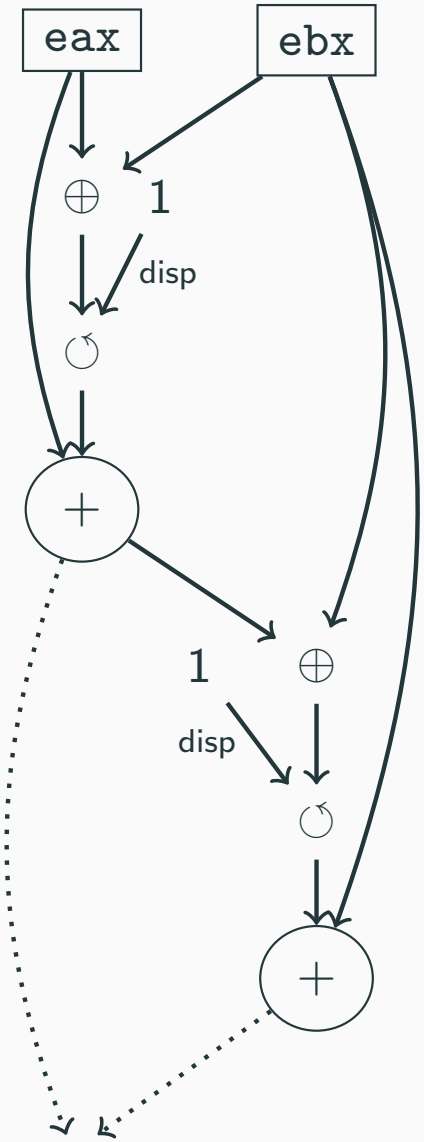
**Recherche:** pour chaque instance $f$ d'une signature $S$ dans un DFG $G$:

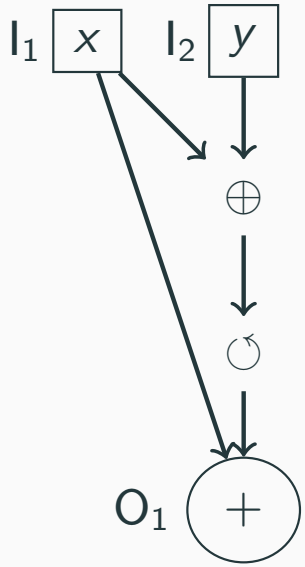
- on ajoute à $G$ un nœud $v$ portant le label $\omega_S$ ;
- pour chaque $x_i \in I_S$ on ajoute à $G$ une arête de $f_V(x_i)$ à $v$ ;
- pour chaque $y_i \in O_S$ on ajoute à $G$ une arête de $v$ à $f_V(y_i)$.

# Exemple - Recherche Signature Composée

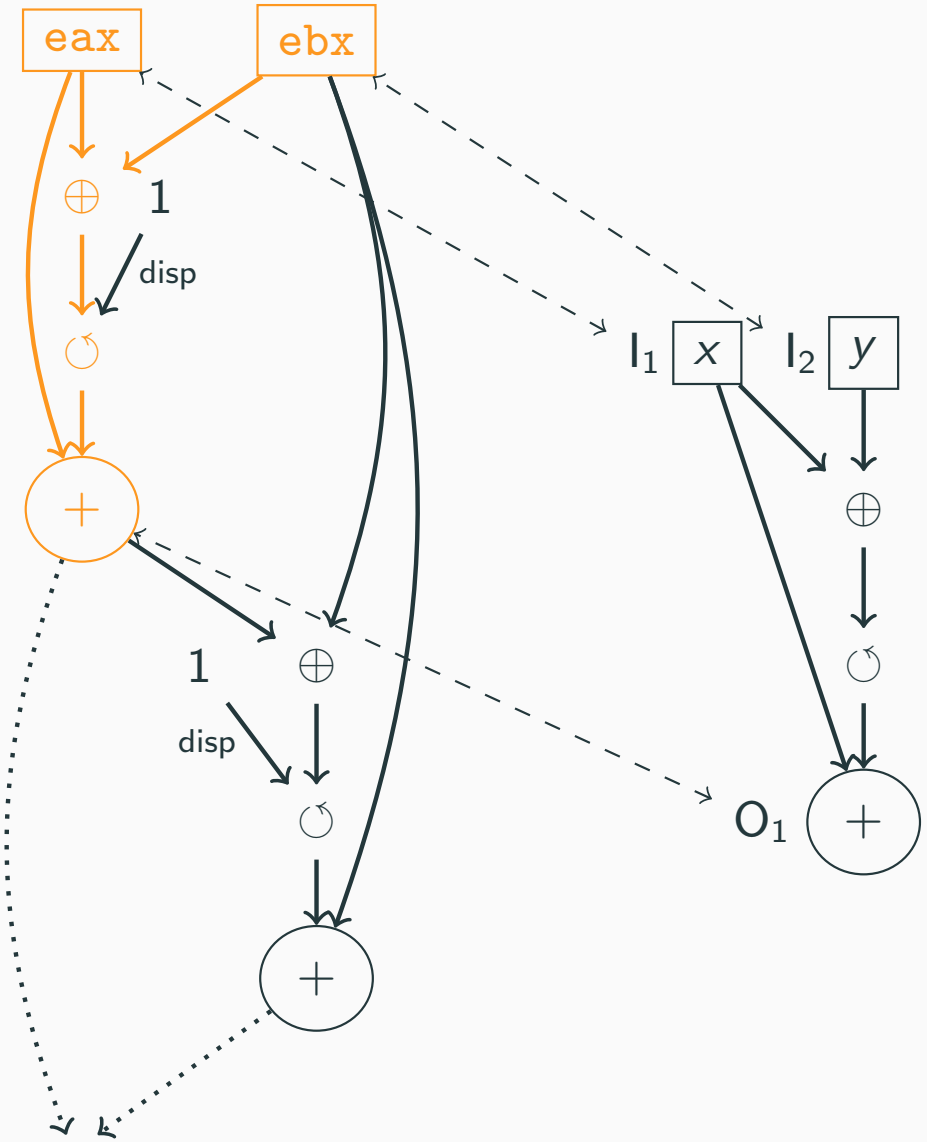


# Exemple - Recherche Signature Composée

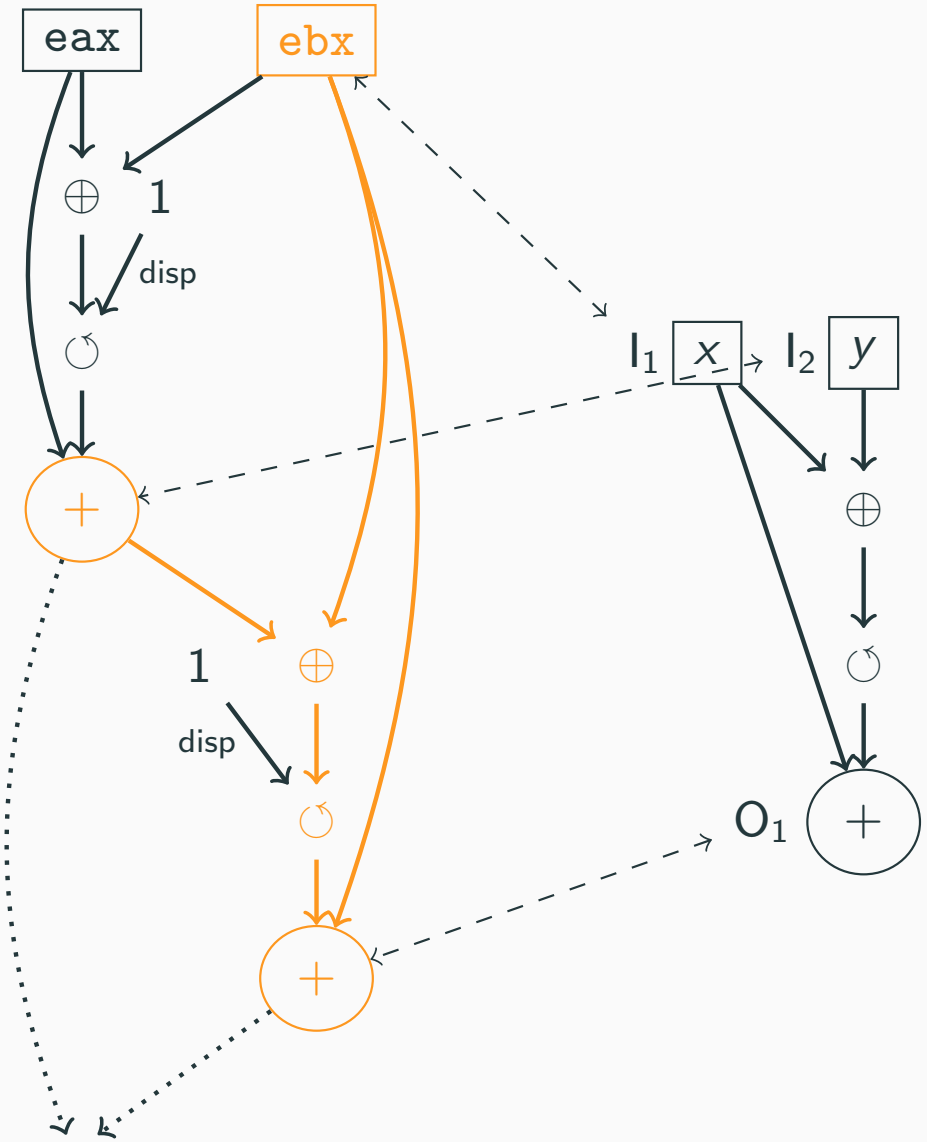


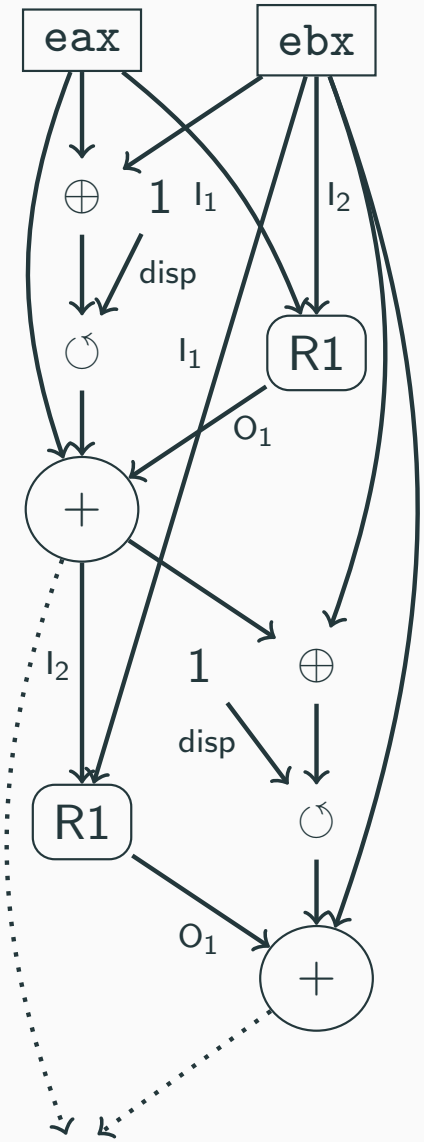
# Exemple - Recherche Signature Composée



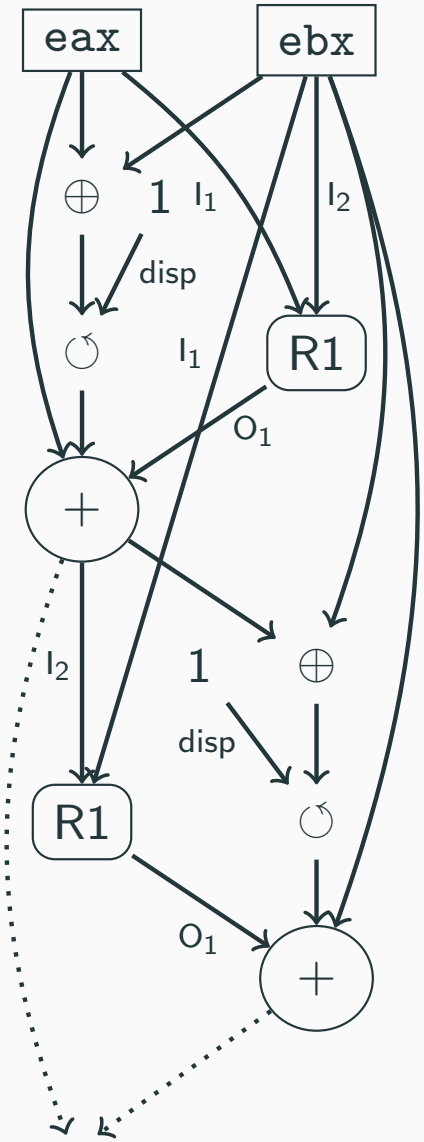
# Exemple - Recherche Signature Composée

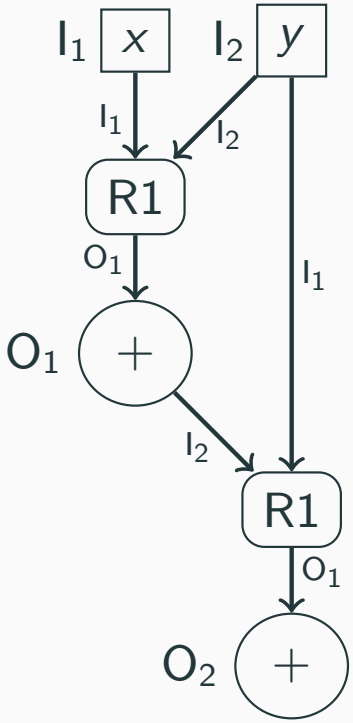


# Exemple - Recherche Signature Composée

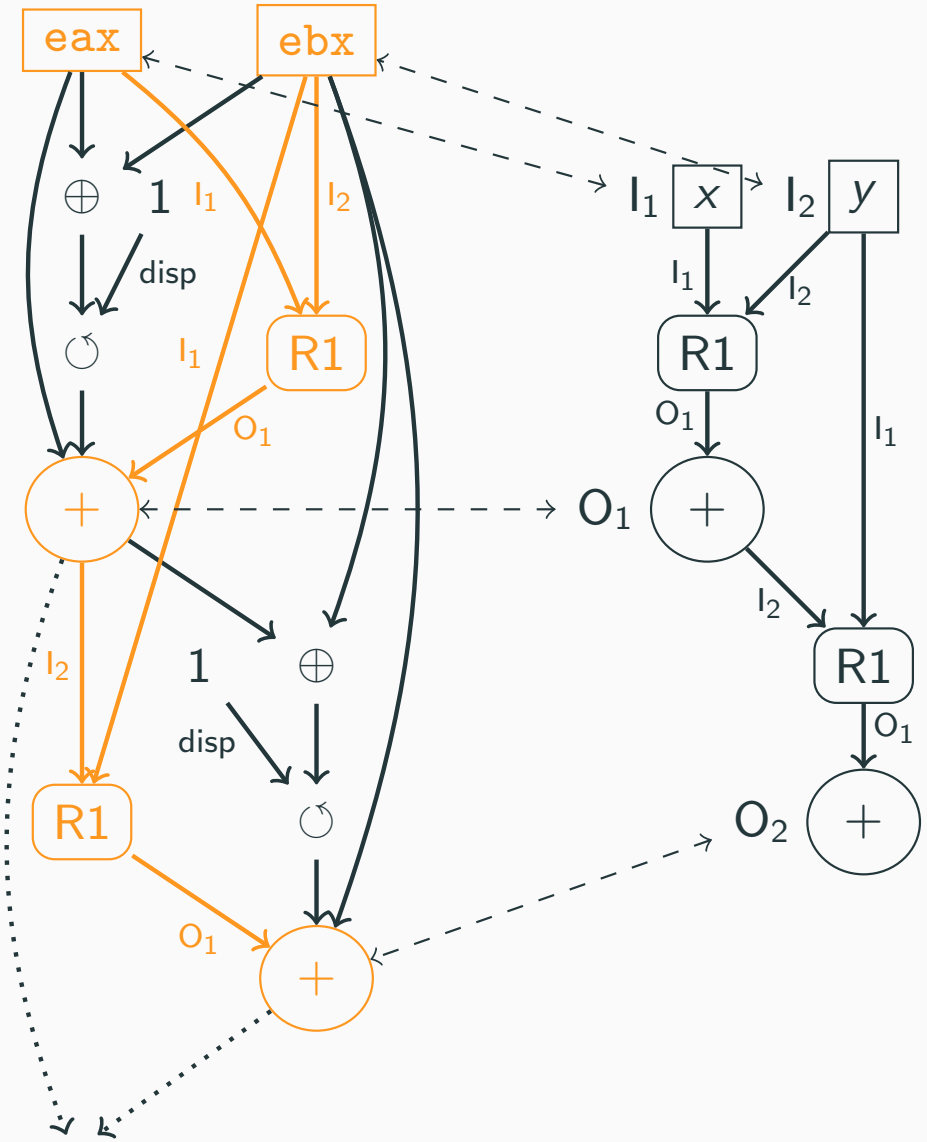


# Exemple - Recherche Signature Composée



# Exemple - Recherche Signature Composée



# Recherche de Signatures Composées

**Problème:** des expressions se retrouvent dupliquées.

**Définitions:**

- $e_1$ est en collision directe avec $e_2$ si $e_2$ résulte de la superposition d'une instance $f$ de $S$ et si $e_1 \in f_E(E_S)$. *(notation $e_1 \lhd e_2$)*
- $e_1$ et $e_2$ sont en collision indirecte si $e_1$ et $e_2$ n'ont pas été ajoutées lors de la superposition d'une même instance et s'il existe $e_3$ telle que $e_3 \lhd^* e_1$ et $e_3 \lhd^* e_2$. *(notation $e_1 \bowtie e_2$)*

**Recherche:**
On ne cherche que les instances ne possédant pas d'arêtes en collision indirecte.

# Évaluation Expérimentale

**Évaluation** sur un ensemble de cas de test

*(différentes primitives, implémentations et conditions de compilation)*

# Ensemble de Signatures

- **AES:** implémentation par tables, tout sauf 1$^{er}$ *addRoundKey*
  A1T4, A1L_V1, . . . , A1L_V8, A10, A12, A14
- **MD5:** fonction de compression
  M1_V1, M1_V2, M2_V1, M2_V2, . . . , M4_V1, M4_V2, MC
- **RC4:** une itération du PRGA
  RP_V1, RP_V2, RP_V3, RP_V4
- **SHA1:** *message schedule*
  SM_V1, SM_V2
- **XTEA:** chiffrement/déchiffrement sauf *key schedule*
  XE1, XE32_V1, XE32_V2, XD1, XD32_V1, XD32_V2

# Détection - Compilations

| | $GCC_{4.9.2}$ | | | | $Clang_{3.5.0}$ | | | | $MSVC_{18.0}$ | |
|---|---|---|---|---|---|---|---|---|---|---|
| | -O0 | -O1 | -O2 | -O3 | -O0 | -O1 | -O2 | -O3 | -O0 | -O2 |
| AES - Gladman $V_1$ | ■ | ■ | ■ | ■ | ■ | ■ | ■ | ■ | ■ | ■ |
| MD5 - Ad hoc | ■ | ■ | ■ | ■ | ■ | ■ | ■ | ■ | ■ | ■ |
| RC4 - Ad hoc | ■ | ■ | ■ | ■ | ■ | ■ | ■ | ■ | ■ | ■ |
| SHA1 - RFC 3174 | ■ | ■ | ■ | ■ | ■ | ■ | ■ | ■ | ■ | ■ |
| XTEA - Wikipedia | ■ | ■ | ■ | ■ | ■ | ■ | ■ | ■ | ■ | ■ |

**Problèmes:**

- MD5: initialisation de la valeur de chaînage contenu dans le segment.
- SHA1: rotation.

# Détection - Bibliothèques

| | AES | MD5 | RC4 | SHA1 | XTEA |
|---|---|---|---|---|---|
| Botan$_{1.10.8}$ | ■ | ■ | ■ | ■ | ■ |
| Crypto++$_{5.6.1}$ | ■ | ■ | ■ | ■ | ■ |
| Nettle$_{2.7.1}$ | ■ | ■ | ■ | ■ | - |
| OpenSSL$_{1.0.1t}$ | ■ | ■ | ■ | ■ | - |
| TomCrypt$_{1.17}$ | ■ | ■ | ■ | ■ | ■ |

**Problèmes:**

- AES - Botan: clé lue octet par octet lors du dernier *addroundkey*.
- AES - Crypto++: API particulière incluant des opérations du mode opératoire.
- AES - OpenSSL: *xtime* calculé explicitement.
- RC4 - Botan: segment trop grand (88k instructions).

# Performance

Exemples issus de la bibliothèque TomCrypt.

| | AES-256 E | MD5 | RC4 | SHA1 | XTEA E |
|---|---|---|---|---|---|
| | *3209 nœuds*<br>*3747 arêtes* | *1327 nœuds*<br>*1801 arêtes* | *663 nœuds*<br>*795 arêtes* | *3333 nœuds*<br>*4267 arêtes* | *771 nœuds*<br>*1029 arêtes* |
| Normalisation | 152 ms | 19 ms | 6 ms | 66 ms | 10 ms |
| | *1693 nœuds*<br>*2154 arêtes* | *802 nœuds*<br>*1206 arêtes* | *425 nœuds*<br>*545 arêtes* | *1354 nœuds*<br>*2139 arêtes* | *741 nœuds*<br>*997 arêtes* |
| Recherche | 495 ms | 38 ms | 1 ms | 59 ms | 41 ms |
| **Total** | **647** ms | **57** ms | **7** ms | **125** ms | **51** ms |

# Conclusion - Identification Primitives

**Inconvénients:**

- implémentations singulières ;
- complexité de la solution: normalisation & signatures.

**Avantages:**

- peu de faux positifs ;
- informations détaillées.

---

**Pistes d'Amélioration:**

- sensibilité au contexte non résolue ;
- ajout de nouveaux mécanismes de normalisation.

# Identification de Modes Opératoires

# Principe - Identification Modes

**Point de départ:**

On peut identifier des primitives, localiser leurs paramètres et les opérations qui les composent.

**Résumer les échanges entre des primitives**

- Étape 1: **Synthèse**
  - Objectif: produire une représentation synthétique.
  - Solution: *slicing* dynamique.
- Étape 2: **Interprétation** - manuelle

# Définition Slice

**Motivation:** critère générique fonctionnant pour tous les modes.

**Définition:**

Soit $G$ un DFG et $PAR$ un ensemble de *paramètres*, une *slice* $\Gamma$ est un **plus petit** sous-graphe de $G$ tel que:

- $P \cap V_\Gamma \neq \emptyset$ pour tous *paramètres* $P \in PAR$
- la distance entre deux *paramètres* $P_1$ et $P_2$ dans $G$ est **égale** à la distance entre $P_1 \cap V_\Gamma$ et $P_2 \cap V_\Gamma$ dans $\Gamma$.

**Remarque:** compromis entre lisibilité et exhaustivité.

# Ajustements Basés sur la Sémantique

**Présupposé:** *(lien implicite entre syntaxe et sémantique)*

$u$ est un prédécesseur de $v \Leftrightarrow u$ *influence* $v$

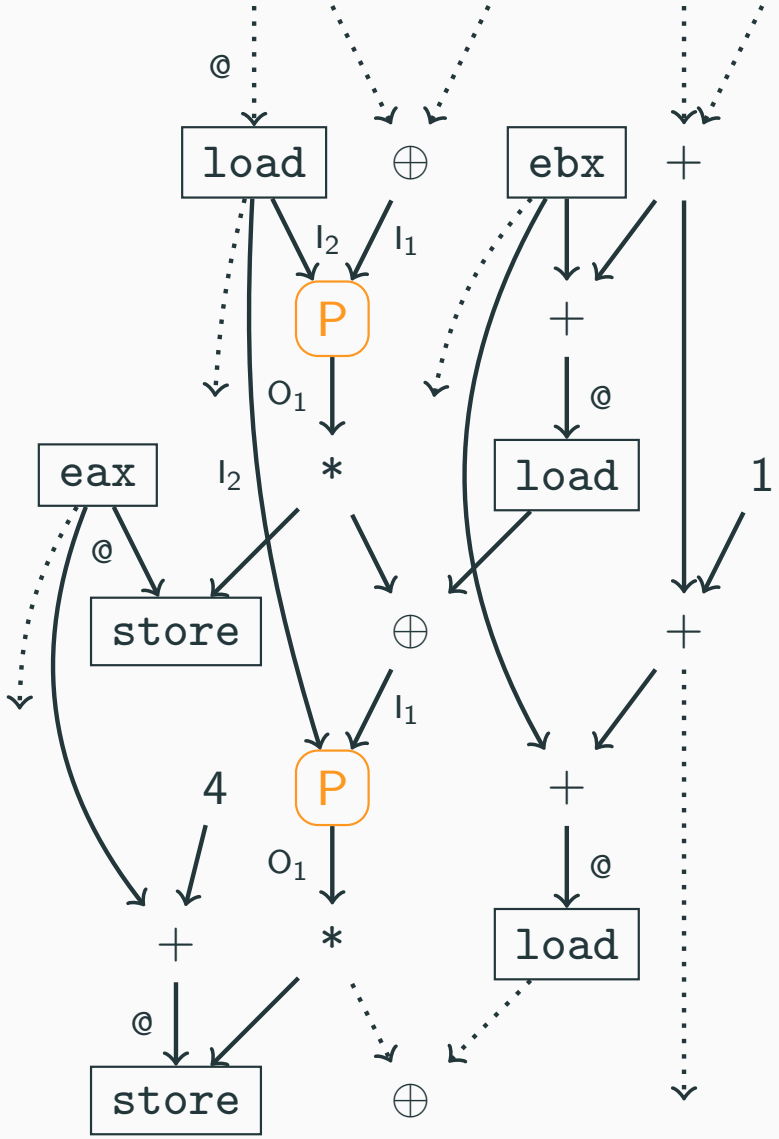
**Ajustements:**

- masques d'influence pour filtrer les chemins ne reflétant pas de réelle influence: $M^{\leftrightarrow}_{u|p}(v) = 00\ldots0$
- simplification des accès mémoire *(valeurs d'adresses concrètes)*.

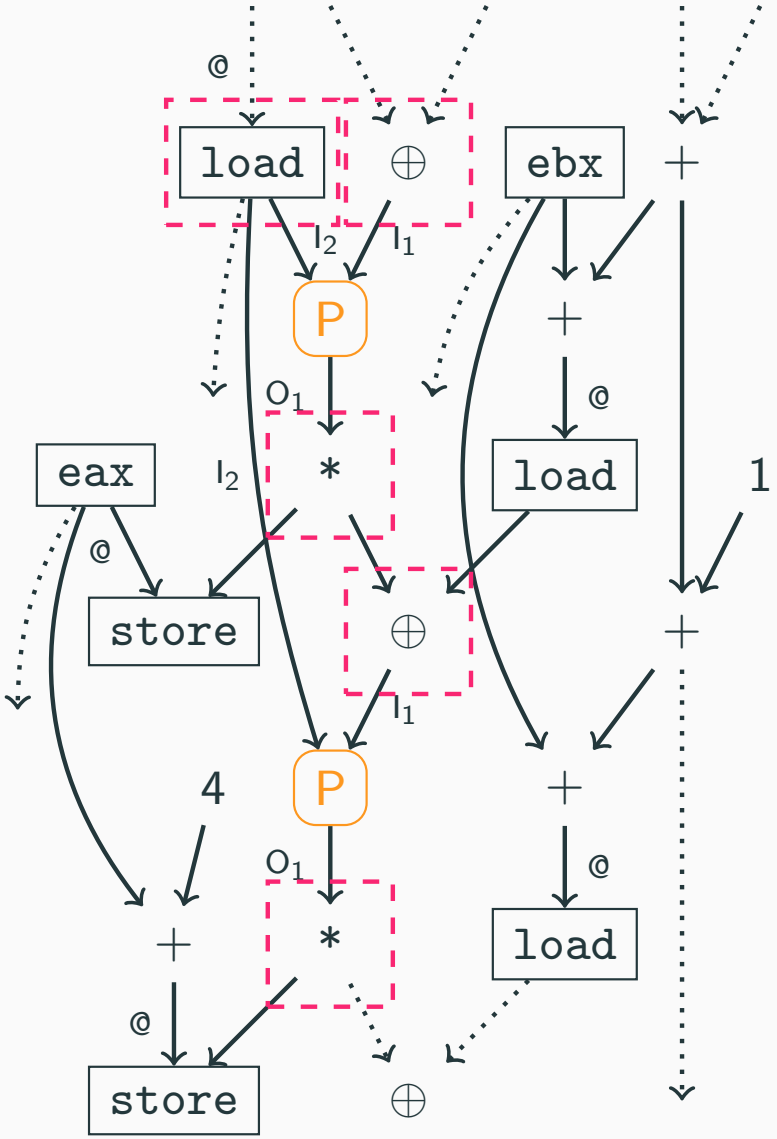
---

- pas de chemin entre un accès mémoire et son adresse.

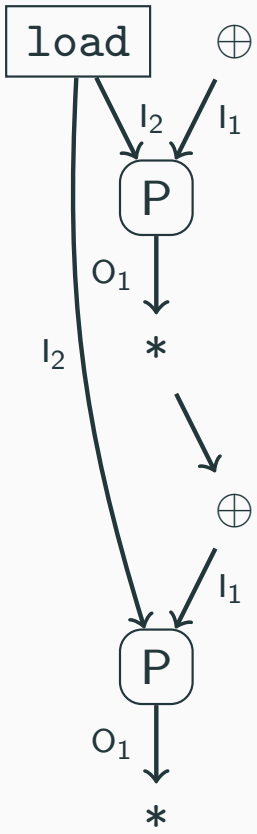
# Exemple - Slicing



# Exemple - Slicing



# Exemple - Slicing



# Extraction d'une Slice

## Méthode:

1. pour chaque paire de *paramètres*, on recherche l'ensemble des plus courts chemins.
2. on sélectionne un plus court chemin par ensemble, tel que leur union soit minimale.

## Remarque:

Algorithme glouton pour résoudre le problème de couverture minimale.
*(sans garantie sur le facteur d'approximation)*

## En Pratique:

- le nombre de plus courts chemins est limité.
- ensembles de plus courts chemins disjoints.

# Évaluation Expérimentale

- Évaluation sur un ensemble de cas de test
  *(différentes modes, implémentations et conditions de compilation)*

- Évaluation sur de nombreux **cas concrets**:
  - logiciel de chiffrement AxCrypt
  - logiciel de messagerie instantanée Telegram
  - générateur aléatoire d'OpenSSL
  - ...

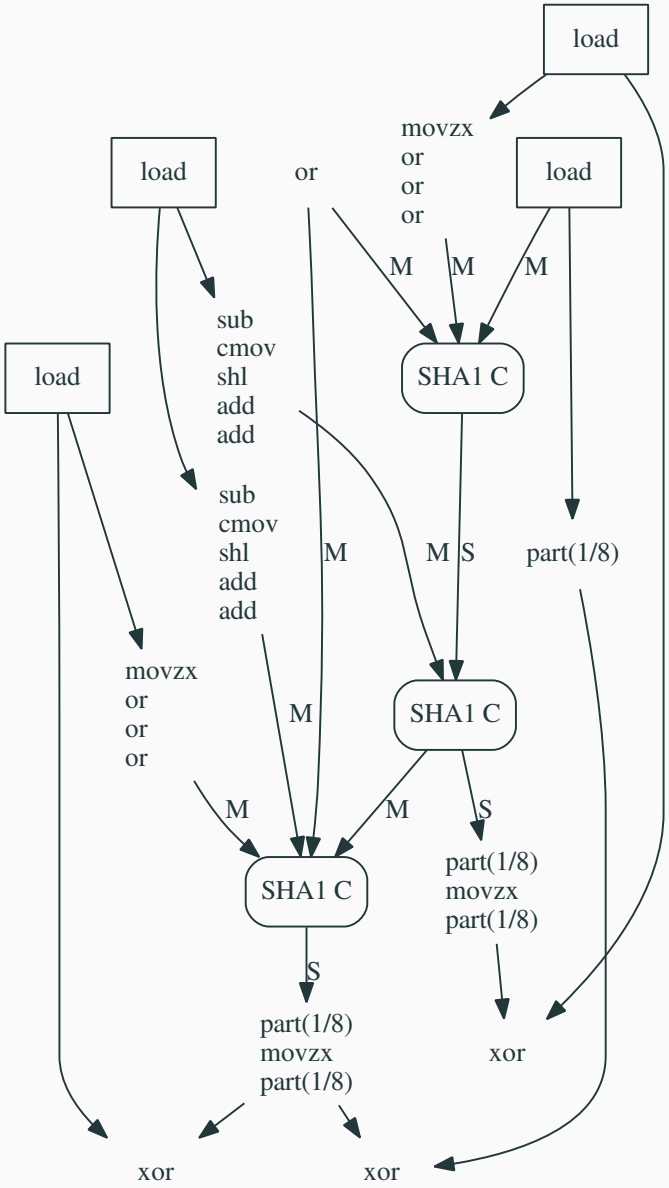
# Cas Concret - OpenSSL RNG

**Rand_add**: incorpore l'entropie d'un buffer $B$ dans l'état interne $S$ du RNG.

**for** $i = 0$ to $n$ **do**
$md_{i+1} \leftarrow SHA1(md_i \,||\, S_i \,||\, B_i \,||\, c_1 \,||\, c_2)$
$S_i \leftarrow S_i \oplus md_{i+1}$
$c_2 \leftarrow c_2 + 1$
**end for**
$md_0 \leftarrow md_0 \oplus md_n$

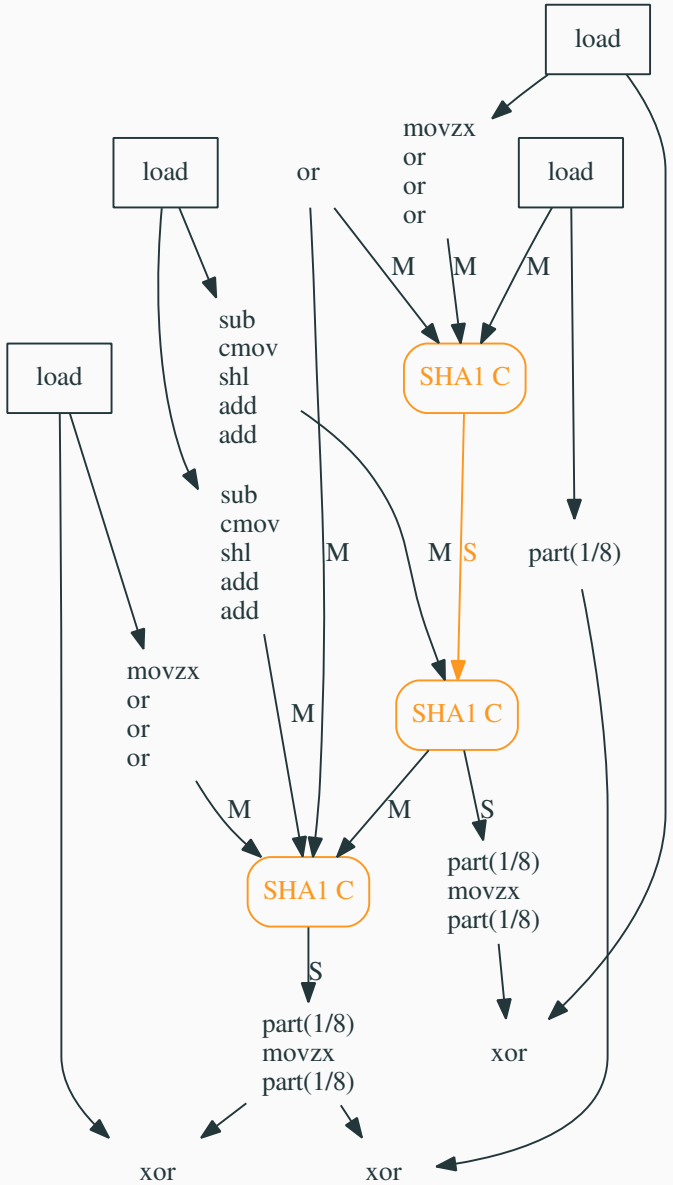
Le buffer d'entrée $B$ et l'état interne $S$ sont divisés en blocs de 20 octets. $c_1$ et $c_2$ sont deux compteurs de 32 bits.

# Cas Concret - OpenSSL RNG



# Cas Concret - OpenSSL RNG

$$md_1 \leftarrow SHA1(md_0 \,||\, S_0 \,||\, B_0 \,||\, c_1 \,||\, c_2)$$
$$S_0 \leftarrow S_0 \oplus md_1$$
$$c_2 \leftarrow c_2 + 1$$
$$md_2 \leftarrow SHA1(md_1 \,||\, S_1 \,||\, B_1 \,||\, c_1 \,||\, c_2)$$
$$S_1 \leftarrow S_1 \oplus md_2$$
$$c_2 \leftarrow c_2 + 1$$
$$md_0 \leftarrow md_0 \oplus md_2$$



# Cas Concret - OpenSSL RNG

$md_1 \leftarrow SHA1(md_0 \,||\, S_0 \,||\, B_0 \,||\, c_1 \,||\, c_2)$

$S_0 \leftarrow S_0 \oplus md_1$

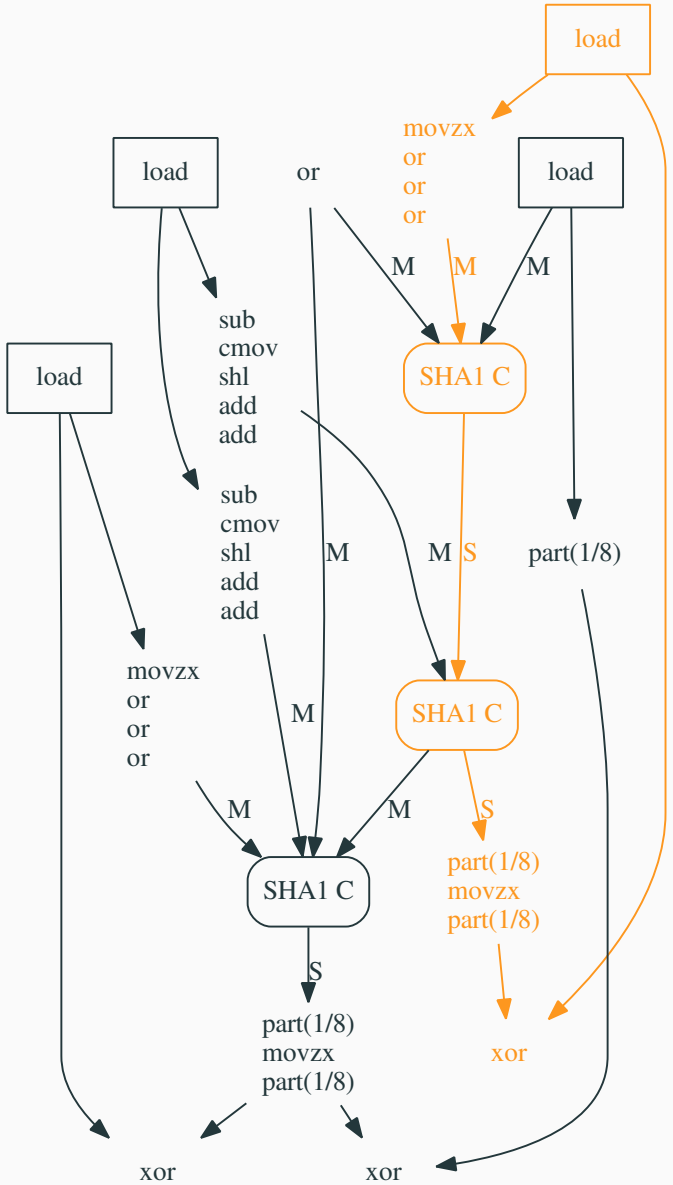
$c_2 \leftarrow c_2 + 1$

$md_2 \leftarrow SHA1(md_1 \,||\, S_1 \,||\, B_1 \,||\, c_1 \,||\, c_2)$

$S_1 \leftarrow S_1 \oplus md_2$

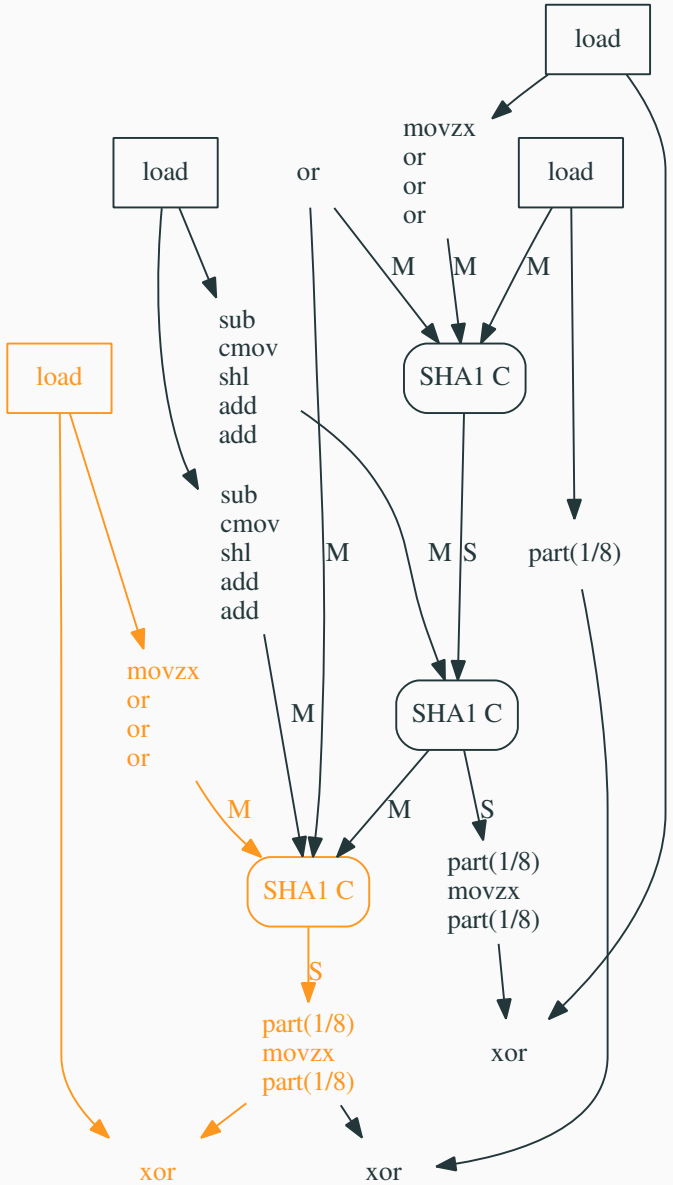
$c_2 \leftarrow c_2 + 1$

$md_0 \leftarrow md_0 \oplus md_2$



# Cas Concret - OpenSSL RNG

$$md_1 \leftarrow SHA1(md_0 \,||\, S_0 \,||\, B_0 \,||\, c_1 \,||\, c_2)$$
$$S_0 \leftarrow S_0 \oplus md_1$$
$$c_2 \leftarrow c_2 + 1$$
$$md_2 \leftarrow SHA1(md_1 \,||\, S_1 \,||\, B_1 \,||\, c_1 \,||\, c_2)$$
$$S_1 \leftarrow S_1 \oplus md_2$$
$$c_2 \leftarrow c_2 + 1$$
$$md_0 \leftarrow md_0 \oplus md_2$$

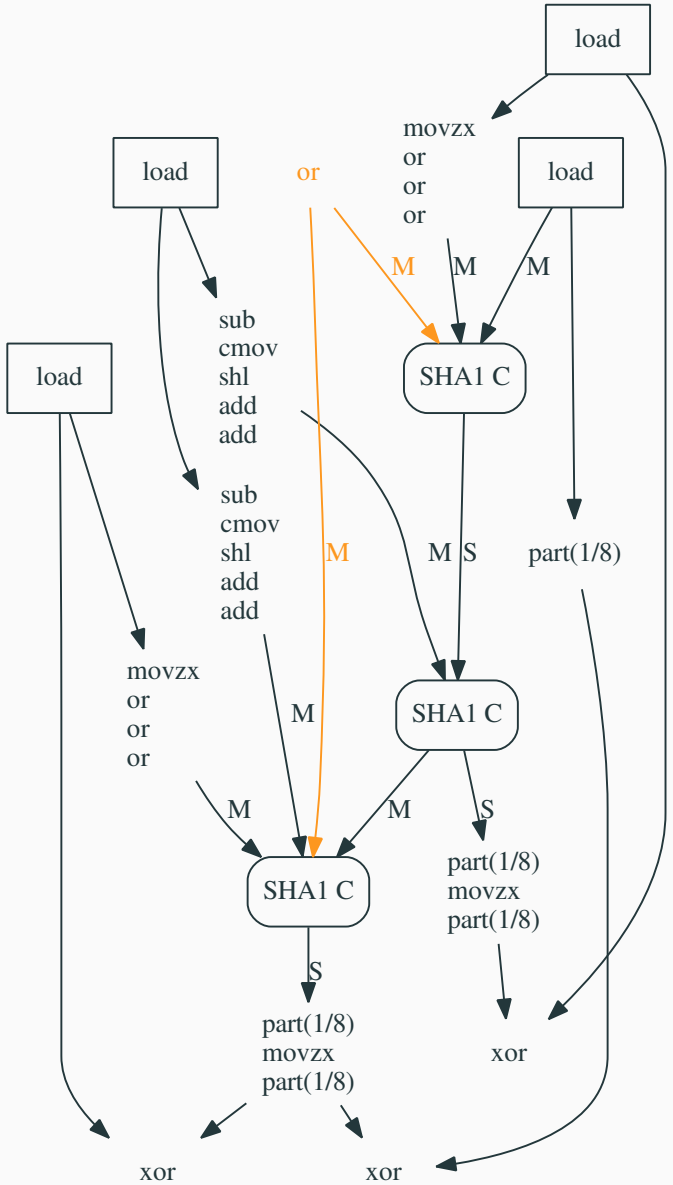


# Cas Concret - OpenSSL RNG

$md_1 \leftarrow SHA1(md_0 \,||\, S_0 \,||\, B_0 \,||\, c_1 \,||\, c_2)$
$S_0 \leftarrow S_0 \oplus md_1$
$c_2 \leftarrow c_2 + 1$

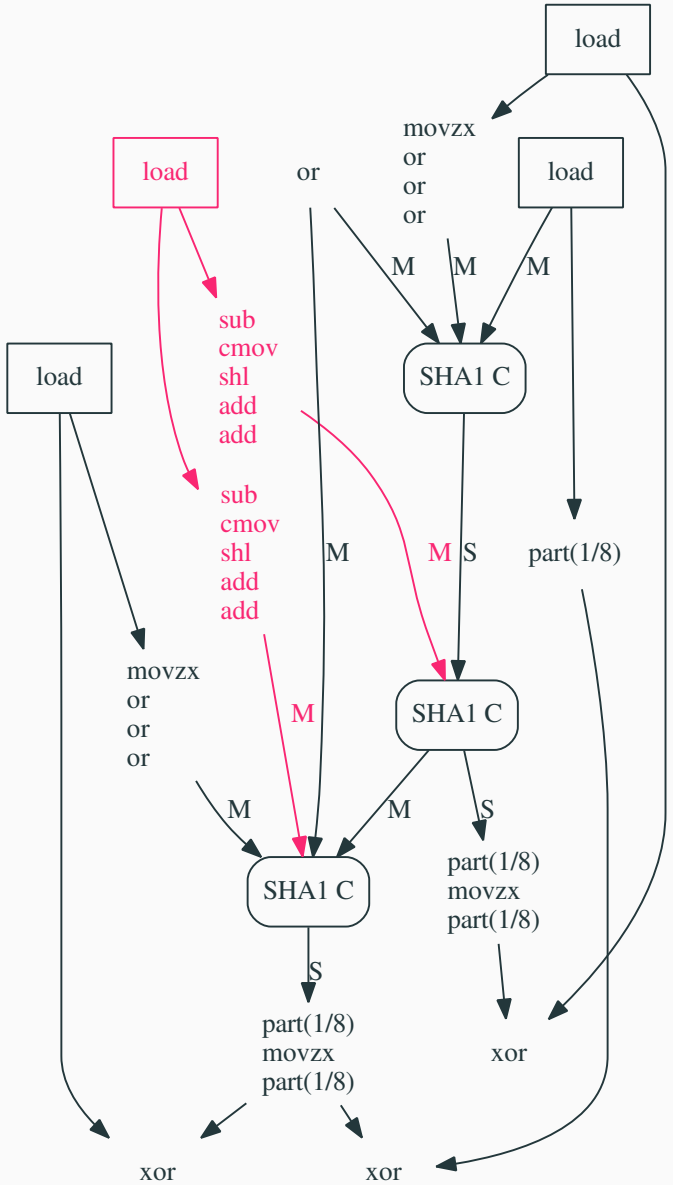
$md_2 \leftarrow SHA1(md_1 \,||\, S_1 \,||\, B_1 \,||\, c_1 \,||\, c_2)$
$S_1 \leftarrow S_1 \oplus md_2$
$c_2 \leftarrow c_2 + 1$

$md_0 \leftarrow md_0 \oplus md_2$



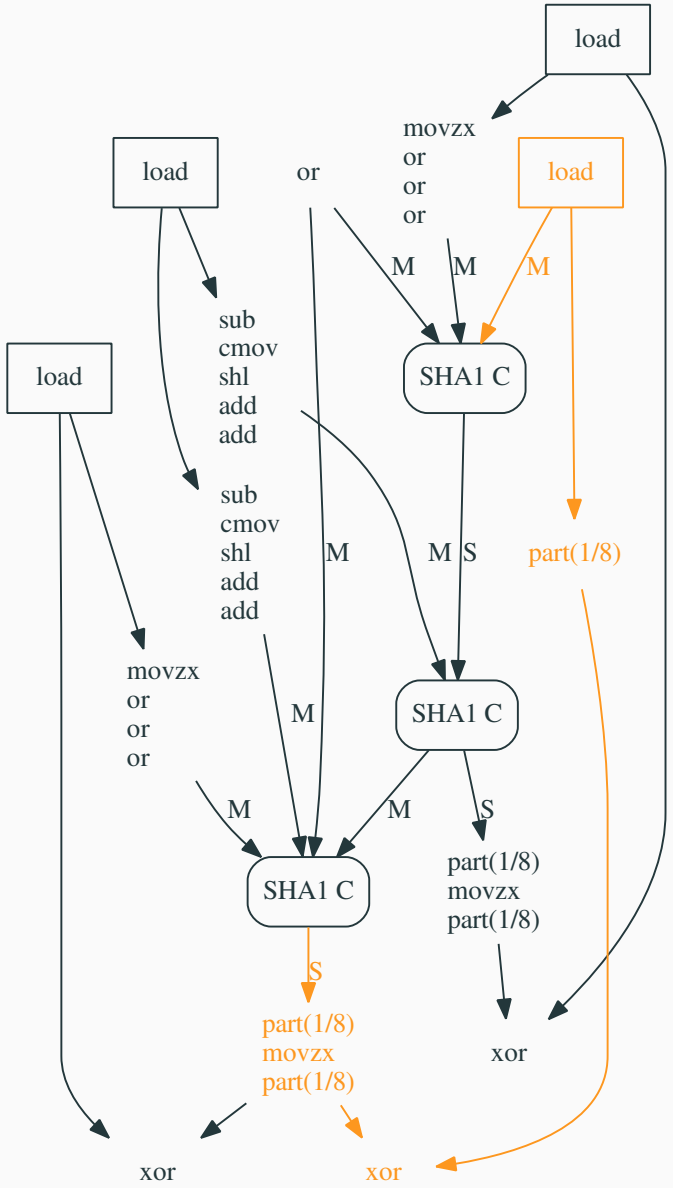
# Cas Concret - OpenSSL RNG

$$md_1 \leftarrow SHA1(md_0 \;||\; S_0 \;||\; B_0 \;||\; c_1 \;||\; c_2)$$
$$S_0 \leftarrow S_0 \oplus md_1$$
$$c_2 \leftarrow c_2 + 1$$
$$md_2 \leftarrow SHA1(md_1 \;||\; S_1 \;||\; B_1 \;||\; c_1 \;||\; c_2)$$
$$S_1 \leftarrow S_1 \oplus md_2$$
$$c_2 \leftarrow c_2 + 1$$
$$md_0 \leftarrow md_0 \oplus md_2$$



# Cas Concret - OpenSSL RNG

$md_1 \leftarrow SHA1(md_0 \,||\, S_0 \,||\, B_0 \,||\, c_1 \,||\, c_2)$
$S_0 \leftarrow S_0 \oplus md_1$
$c_2 \leftarrow c_2 + 1$
$md_2 \leftarrow SHA1(md_1 \,||\, S_1 \,||\, B_1 \,||\, c_1 \,||\, c_2)$
$S_1 \leftarrow S_1 \oplus md_2$
$c_2 \leftarrow c_2 + 1$
$md_0 \leftarrow md_0 \oplus md_2$



# Conclusion - Identification Modes

**Quelques Défauts:**

- relations peu pertinentes reportées dans certains cas *(ex: HMAC)*
- prédominance des plus courts chemins pas toujours opportune.
- modes complexes $\rightarrow$ *slices* difficiles à interpréter.
- informations peu détaillées et aucunes garanties (approximatif)

Une démarche de synthèse d'information qui diffère des signatures.

---

**Pistes d'Amélioration:**

- nouveau critère de *slicing* ;
- faire figurer d'avantage d'information.

# Questions?

# Conclusion

## Résumé et Contributions:

- Identification de primitives par normalisation et recherche d'expressions

**Automated Identification of Cryptographic Primitives in Binary Code with Data Flow Graph Isomorphism**
*Pierre Lestringant, Frédéric Guihéry, Pierre-Alain Fouque* - ASIA CCS 2015

- Identification de modes opératoires par *slicing* dynamique

**Assisted Identification of Mode of Operation in Binary Code with Dynamic Data Flow Slicing**
*Pierre Lestringant, Frédéric Guihéry, Pierre-Alain Fouque* - ACNS 2016

Implémentation et cas de test disponibles à `plestrin/bacs`

# Hypothèses de Départ

**Hypothèses:**

- Accès au code machine et possibilité de l'exécuter.

  *($\neq$ d'une situation black box)*

- Localisation approchée des algorithmes cryptographiques.

  *(exemples: constantes et mnémoniques spécifiques, taille des basic blocs, nombre d'exécutions, position dans l'arbre d'appel, . . . )*

- Code non obfuscé.

  *(analyse dynamique peu impactée par le packing)*

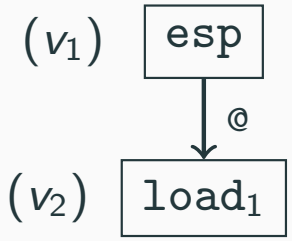
# Construction

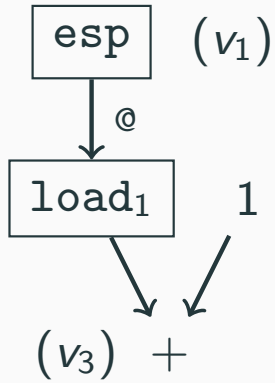
Construction par combinaison d'instructions.

**Exemple:**

```
mov eax,[esp]
```

| État Registres | | |
|---|---|---|
| esp | → | $v_1$ |
| eax | → | $v_2$ |



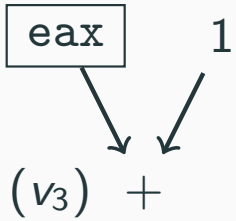




```
mov al,[esp]
add eax,1
```

| État Registres | | |
|---|---|---|
| esp | → | $v_1$ |
| eax | → | $v_3$ |

```
add eax,1
```

| État Registres | | |
|---|---|---|
| eax | → | $v_3$ |




**Remarque:** *état des registres* mis à jour durant la normalisation, ce qui permet l'élargissement du segment d'analyse.

# Instructions SIMD

**Observation:** l'usage d'instructions *SIMD* est fréquent dans les codes cryptographiques.

Par anticipation de la normalisation, on souhaite s'en abstraire.

- **Division d'Instructions:** partage en plusieurs instructions de taille identique. Difficulté: trouver la bonne taille.

  *Exemples:* `paddb`, `paddd`, `pshlld`, `pand`, `pxor`, `xorps`, `xorpd`, ...

- **Évaluation Partielle:** prise en compte d'opérande(s) immédiate(s) pour simplifier la sémantique du mnémonique.

  *Exemples:* `palignr`, `pinsrw`, `pshufd`, `pslldq`, ...

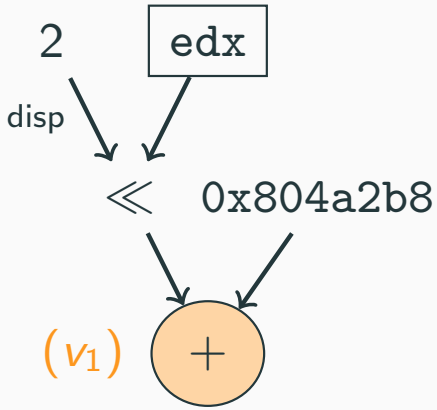
# Outils - Domaine de Variations

**Motivation:** déterminer l'ensemble formé par les différentes évaluations d'un nœud.

**Solution:** sur-approximation par un ensemble de la forme:

$$\{2^a x + c \pmod{2^d}, x \in \mathbb{N}, x \leq b\}$$

**Exemple:**




$$D_G(v_1) = \{2^2 x + \texttt{0x804a2b8} \pmod{2^{32}}, x \in \mathbb{N}, x \leq 2^{32} - 1\}$$

**Remarque:** adapté aux opérations arithmétiques modulaires mais pas aux opérations logiques bit à bit.

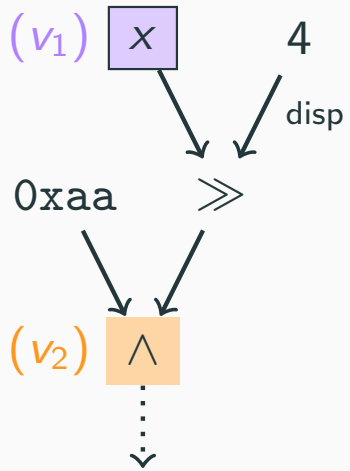
# Outils - Masque d'Influence

**Motivation:** caractériser au bit près, pour un nœud $v$, l'influence qu'ont sur lui ses prédécesseurs.

## Définition:

Le $i^{\text{ième}}$ bit de $M_v^{\leftarrow}(u)$ vaut: 0 **si** pour toute évaluation $\theta$, $\theta_G(v)$ reste inchangé lorsque l'on inverse le $i^{\text{ième}}$ bit de $\theta_G(u)$; 1 **sinon**.

## Exemple:




$$M_{v_2}^{\leftarrow}(v_1) = 01010000$$

# Formalisation - Étape de Réécriture

**Définition:**

Une règle de réécriture est un triplet $(L, R, \psi)$, où:

- $L$ est un DFG ;
- $R$ est un DFG possédant les mêmes variables que $L$ ;
- $\psi$ est une fonction surjective de $Root_L$ vers $Root_R$.

**Définition:**

Soient $G$ et $H$ deux DFGs et $r = (L, R, \psi)$ une règle de réécriture. Il existe une étape de réécriture entre $G$ et $H$ si $H$ s'obtient à partir de $G$ en *remplaçant* un sous-graphe de $G$ isomorphe à $L$ par $R$.

*(notation $G \rightarrow_{r,f} H$)*

# Mécanismes de Normalisation

## Constant Folding

**Objectif:** simplification d'expressions.

**Règles:** *(exprimées ici pour $\times$, autres opérations: $+$, AND, OR, XOR, shift)*

$$\times(0, x_2, ..., x_n) \to 0$$
$$\times(1, x_2, ..., x_n) \to \times(x_2, ..., x_n)$$
$$\times(c_1, ..., c_n) \to c_1 \times ... \times c_n$$
$$\times(c_1, ..., c_j, x_{j+1}, ..., x_n) \to \times(c_1 \times ... \times c_j, x_{j+1}, ..., x_n)$$

**Implémentation:** en une seule passe lorsque les nœuds sont triés suivant un ordre topologique.

**Remarque:** accroît fortement la sensibilité au contexte.
(*Exemple:* initialisation de la valeur de chaînage pour MD5 et SHA1.)

## Distribution des Opérations avec Opérandes Constantes

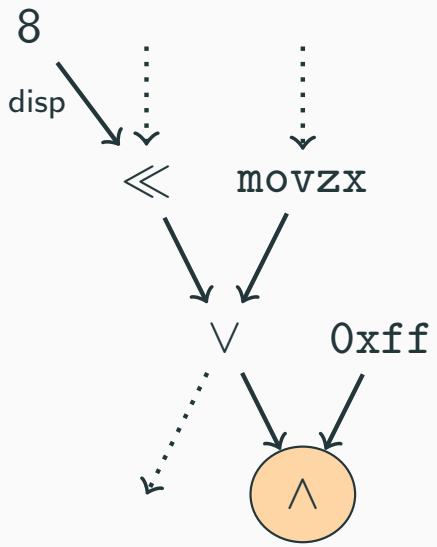
**Objectif:** augmenter les possibilités de *constant folding*.

**Règle:** *(exprimée ici pour* $(\times, +)$*, autres opérations:* $(\ll, +)$, $(\ll, \vee)$, $(\gg, \vee)$, $(\ll, \wedge)$, $(\gg, \wedge)$, $(\wedge, \vee)$*)*

Si $D_G(\times(c_1, x_1))$ est un singleton:

$$\times(c_1, +(x_1, x_2)) \rightarrow +(\times(c_1, x_1), \times(c_1, x_2))$$

**Exemple:**



# Mécanismes de Normalisation

## Fusion des Opérations avec Opérandes Constantes

**Objectif:** augmenter les possibilités de *constant folding*.

**Règle:** *(exprimée ici pour $+$, autre opération $\wedge$)*

$$+(c_1, +(\ldots, c_2, \ldots), \ldots) \rightarrow +(c_1, c_2, \ldots)$$

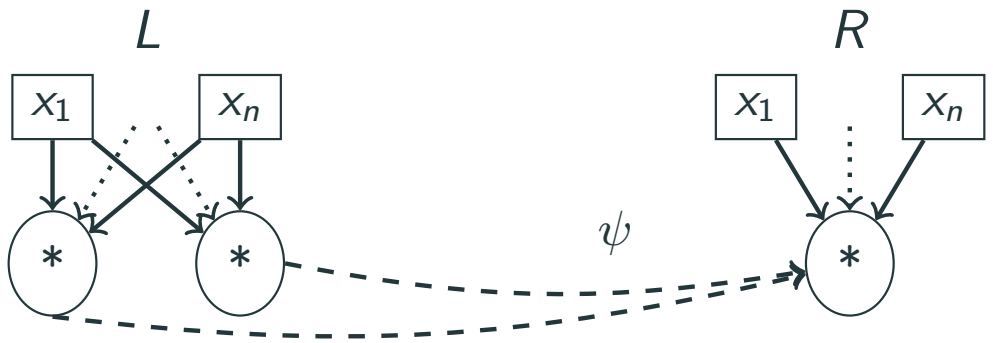
**Remarques:**

- conflit avec suppression des sous-expressions communes.
- similarité avec normalisation des opérations commutatives & associatives.

# Mécanismes de Normalisation

## Suppression des Sous-Expressions Communes

**Règle:** *(toutes les opérations sauf `load` et `store`)*




**Implémentation:** prise en compte de chaque paire de nœuds, ceux-ci devant être triés suivant un ordre topologique.

**Remarque:** règles additionnelles ne concernant que les opérations commutatives & associatives.

# Mécanismes de Normalisation

## Elargissement d'Expressions

```
; Extrait 1
add al, bl
movzx eax, al
```

```
; Extrait 2
add eax, ebx
and eax, 0xff
```

## Règle:

Augmenter la taille des opérations lorsqu'elle est inférieure à 32 bits.

## Implémentation: *(3 passes exécutées une seule fois)*

- Augmentation de la taille de: $+$, $\wedge$, `cmov`, `neg`, $\neg$, $\vee$, $\ll$, $-$, $\oplus$
- Suppression des opérations de modification de taille non nécessaires.
- Insertion de nouvelles opérations de modification de taille.

**Remarque:** pas de masquage des opérandes.

# Mécanismes de Normalisation

## Memory Coalescing

```
; Extrait 1
mov al, [esp + 1]
shl ax, 8
mov al, [esp]
```

```
; Extrait 2
mov ax, [esp]
```

## Règle:

Regrouper les opérations mémoire adjacentes en une seule opération.

## Remarques:

- ordonnancer les nouvelles opérations.
- tentative pour aligner les adresses.

# Mécanismes de Normalisation

## Simplification des Opérations Affines

$$+(\ldots,\ \times(c_1, x),\ \ldots,\ \times(c_2, x),\ \ldots)$$

**Règle:**

Réécrire les expressions affines sous la forme:

$$+(c_0,\ \times(c_1, x_1),\ \ldots,\ \times(c_n, x_n))$$

**Implémentation:**

- recherche d'expressions affines de taille maximale ;
- réécriture si simplification possible.

**Utilisation:** adresses mémoire.

# Mécanismes de Normalisation

## Normalisation des Opérations Commutatives & Associatives

$$+(+(x_1, x_2), +(x_3, x_4))$$
$$+(+(+(x_1, x_2), x_3), x_4)$$

**Règle:** *(exprimée ici pour $+$, autres opérations: $\vee$, $\oplus$)*

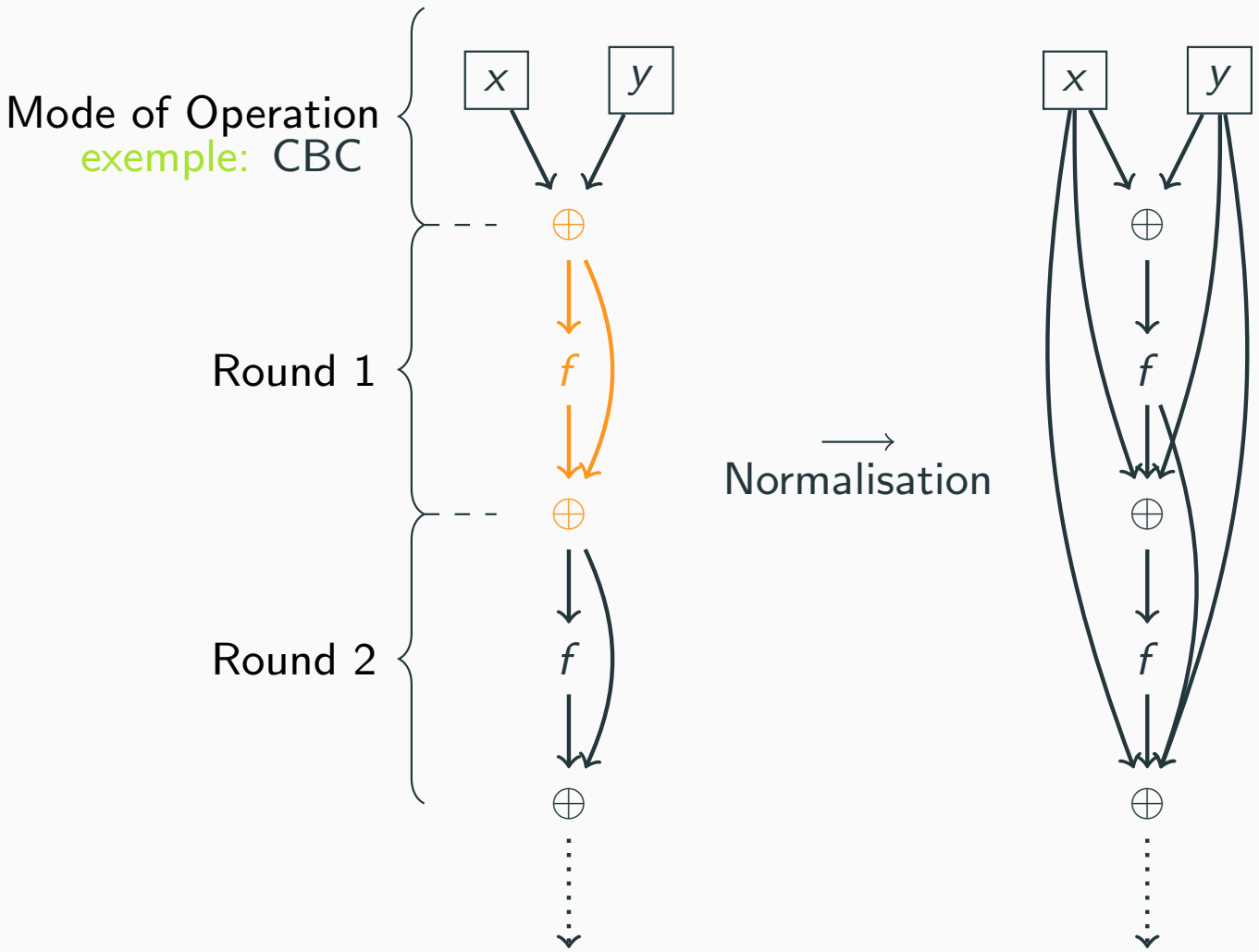
$$+(+(x_1, \ldots, x_n), y_1, \ldots, y_m) \rightarrow +(x_1, \ldots, x_n, y_1, \ldots, y_m)$$

Forte sensibilité au contexte en pratique *(voir exemple)*

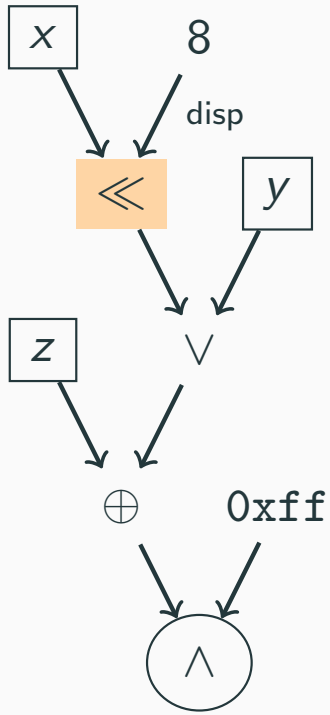
**Atténuations:**

- $u$ ne doit pas avoir d'autre successeur que $v$ *(pas toujours possible)*
- $u$ et $v$ doivent appartenir à une même fonction.

# Normalisation des Opérations Commutatives & Associatives



## Détection d'Expressions Constantes




(extrait du cas de test OpenSSL AES)

**Règle:**

Remplacer le label d'un nœud $v$ par la constante $c$ lorsque: $D_G(v) \wedge M^{\leftarrow}_{Root_G} = \{c\}$

**Implémentation:** en une seule passe lorsque les nœuds sont triés suivant un ordre topologique *(hors mise à jour des domaines de variations)*.

**Remarque:** utilisé principalement en présence d'instructions *SIMD*.

# Mécanismes de Normalisation

## Simplification des Accès Mémoire

**Objectif:** s'abstraire du stockage des variables en mémoire.

**Règles:**

Les adresses des accès $i$ et $j$ sont égales pour toute évaluation:

- s'il n'existe pas de `load` *alias* entre $i$ et $j$:

$$\ldots, \texttt{store}_i, \ldots, \texttt{store}_j \rightarrow \ldots, \texttt{store}_j, \ldots$$

- s'il n'existe pas de `store` *alias* entre $i$ et $j$:

$$\ldots, \texttt{store}_i, \ldots, \texttt{load}_j \rightarrow \ldots, \texttt{store}_i, \ldots$$

- s'il n'existe pas de `store` *alias* entre $i$ et $j$:

$$\ldots, \texttt{load}_i, \ldots, \texttt{load}_j \rightarrow \ldots, \texttt{load}_i, \ldots$$

Deux pointeurs $p_1$ et $p_2$ sont *alias* s'il existe $\theta$ tel que: $\theta_G(p_1) = \theta_G(p_2)$

# Mécanismes de Normalisation

## Méthodes de Comparaison d'Adresses

### Statique:

- Sûre: approximation de la différence entre $p_1$ et $p_2$. Insuffisante
- Approchée: hypothèse sur `esp`. Limitée et incorrecte

### Dynamique: *(valeurs d'adresses concrètes)*

- Entièrement dynamique (incluant *must alias*). Signatures impossibles *(exemple: SBox)*
- Partiellement dynamique (seulement *may*/*cannot alias*). Incorrecte