USUBA v2, Syntaxe et Sémantique

Samuel VIVIEN, sous l'encadrement de Pierre-Évariste DAGAND

3 septembre 2023

```
table SubColumn (input:v4) returns (out:v4)
    { 6, 5, 12, 10, 1, 14, 7, 9, 11, 0, 3, 13, 8, 15, 4, 2 }

node ShiftRows (input:u16[4]) returns (out:u16[4])
vars
let
    out[0] = input[0];
    out[1] = input[1] <<< 1;
    out[2] = input[2] <<< 12;
    out[3] = input[3] <<< 13
tel

node Rectangle (plain:u16[4], key:u16[26][4])
returns (cipher:u16[4])
vars tmp : u16[26][4]
let
    tmp[0] = plain;
    forall i in [0,24] {
        tmp[i+1] = ShiftRows( SubColumn( tmp[i] ^ key[i] ) )
    }
    cipher = tmp[25] ^ key[25]
tel
```

Spécificités du langage

- Pas de conditionnelles (`if` ou `while`)
- Pas d'accès mémoire dynamiques

Spécificités du langage

- Pas de conditionnelles (`if` ou `while`)
- Pas d'accès mémoire dynamiques

Interdit pour avoir un temps d'exécution indépendant des valeurs du calcul

Limitations du langage

- Pas de système de type
- Pas de spécification de la sémantique
- La sémantique implémentées est non compositionnelle !

Limitations du langage

- Pas de système de type
- Pas de spécification de la sémantique
- La sémantique implémentées est non compositionnelle !

On prend $v = [[0, 1], [2, 3], [4, 5]]$
Donc $v[0, 1][1] = [1, 3]$
On prend $\{x = v[0, 1];\ y = x[1]\}$, on obtient donc $y = [2, 3]$

Limitations du langage

- Pas de système de type
- Pas de spécification de la sémantique
- La sémantique implémentées est non compositionnelle !

On prend $v = [[0, 1], [2, 3], [4, 5]]$
Donc $v[0, 1][1] = [1, 3]$
On prend $\{x = v[0, 1];\ y = x[1]\}$, on obtient donc $y = [2, 3]$
Correction : distinguer $v[0, 1;\ 1]$ de $v[0, 1][1]$

Amélioration des appels

```
node MapRectangle (plain:u16[64][4], key:u16[64][26][4])
returns (cipher:u16[64][4])
vars
let
    forall i in [0,63] {
        chiper[i] = Rectangle( plain[i], key[i] )
    }
tel
```

Amélioration des appels

```
node MapRectangle (plain:u16[64][4], key:u16[64][26][4])
returns (cipher:u16[64][4])
vars
let
    forall i in [0,63] {
        chiper[i] = Rectangle( plain[i], key[i] )
    }
tel

node MapRectangle (plain:u16[64][4], key:u16[64][26][4])
returns (cipher:u16[64][4])
vars
let
    chiper = Rectangle[64]( plain, key )
tel
```

Grammaire des types

$$
\begin{array}{l}
dir ::= \\
\quad \mid \mathbf{V} \\
\quad \mid \mathbf{H} \\
\quad \mid d
\end{array}
\qquad
\begin{array}{l}
size ::= \\
\quad \mid s \\
\quad \mid z
\end{array}
$$

$$
\begin{array}{l}
\tau ::= \\
\quad \mid \mathbf{U}\ dir\ size \\
\quad \mid \tau[a]
\end{array}
$$

U V 32 : entier 32 bits classique.

U H 64 : entier 64 bits découpé dans 64 registres.

$u32 = \mathbf{U}\ dir\ 32$ et $v4 = \mathbf{U}\ dir\ size[4]$.

Grammaire des types

$$
\begin{array}{l}
dir ::= \\
\quad \mid \mathbf{V} \\
\quad \mid \mathbf{H} \\
\quad \mid d
\end{array}
\qquad
\begin{array}{l}
\tau ::= \\
\quad \mid \mathbf{U}\ dir\ size \\
\quad \mid \tau[a]
\end{array}
$$

$$
\begin{array}{l}
size ::= \\
\quad \mid s \\
\quad \mid z
\end{array}
\qquad
\begin{array}{l}
\mathcal{T} ::= \\
\quad \mid \overline{\tau_n}
\end{array}
$$

Règles de typage

$$\frac{\Gamma, P, A \vdash_E e_1 : \tau \qquad \Gamma, P, A \vdash_E e_2 : \tau}{\Gamma, P, A \vdash_E e_1 +_\tau e_2 : \tau} \text{ BINOP}$$

Règles de typage

$$\frac{\begin{array}{c}\Gamma, P, A \vdash_E e_1 : \tau \\ \Gamma, P, A \vdash_E e_2 : \tau \\ A \vdash \textbf{Arith}\,\tau\end{array}}{\Gamma, P, A \vdash_E e_1 +_\tau e_2 : \tau}\ \textsc{Binop}$$

Règles de typage

$$\frac{\Gamma, P, A \vdash_E e_1 : \tau \quad \Gamma, P, A \vdash_E e_2 : \tau \quad A \vdash \textbf{Arith}\,\tau}{\Gamma, P, A \vdash_E e_1 +_\tau e_2 : \tau} \text{ BINOP}$$

$$\frac{\Gamma, P, A \vdash_E \overline{e_n : \mathcal{T}_n}}{\Gamma, P, A \vdash_E (\overline{e_n}) : \overline{\mathcal{T}_n}} \text{ TUPLE}$$

Règles de typage des appels de nœuds

$$\frac{P \vdash f : \overline{typc_j} \Rightarrow \mathcal{T}_1 \to \mathcal{T}_2 \qquad \Gamma, P, A \vdash_E (\overline{e_n}) : \sigma_x \overline{[z_q]} \qquad A \vdash \overline{typc_j} \qquad \mathcal{T}_1 = \sigma_x \overline{[z_q]} \qquad \mathcal{T}_2 = \sigma'_y \overline{[z'_r]}}{\Gamma, P, A \vdash_E f (\overline{e_n}) : \sigma'_y \overline{[z'_r]}} \text{ FUN}$$

Règles de typage des appels de nœuds

$$\frac{\begin{array}{c} P \vdash f : \forall\, \overline{d_n}\,, \forall\, \overline{s_m}\,, \overline{typc_j} \Rightarrow \mathcal{T}_1 \to \mathcal{T}_2 \\ \Gamma, P, A \vdash_E (\,\overline{e_n}\,) : \sigma_x\, \overline{[z_q]} \\ A \vdash \overline{typc_j[\,\overline{d_n \leftarrow d'_n}\;;\;\overline{s_m \leftarrow s'_m}\,]} \\ \mathcal{T}_1[\,\overline{d_n \leftarrow d'_n}\;;\;\overline{s_m \leftarrow s'_m}\,] = \sigma_x\, \overline{[z_q]} \\ \mathcal{T}_2[\,\overline{d_n \leftarrow d'_n}\;;\;\overline{s_m \leftarrow s'_m}\,] = \sigma'_y\, \overline{[z'_r]} \end{array}}{\Gamma, P, A \vdash_E f\,(\,\overline{e_n}\,) : \sigma'_y\, \overline{[z'_r]}}\ \textsc{Fun}$$

Règles de typage des appels de nœuds

$$\frac{\begin{array}{c} P \vdash f : \forall\, \overline{d_n}\,, \forall\, \overline{s_m}\,, \overline{typc_j} \Rightarrow \mathcal{T}_1 \to \mathcal{T}_2 \\ \Gamma, P, A \vdash_E (\,\overline{e_n}\,) : \mathcal{T}_1' \\ A \vdash \overline{typc_j[\,\overline{d_n} \leftarrow \overline{d_n'}\;;\;\overline{s_m} \leftarrow \overline{s_m'}\,]} \\ \mathcal{T}_1[\,\overline{d_n} \leftarrow \overline{d_n'}\;;\;\overline{s_m} \leftarrow \overline{s_m'}\,] = \sigma_x\ \overline{\overline{[z_q]}} \\ \mathcal{T}_2[\,\overline{d_n} \leftarrow \overline{d_n'}\;;\;\overline{s_m} \leftarrow \overline{s_m'}\,] = \sigma_y'\ \overline{\overline{[z_r']}} \\ \mathcal{T}_1' \cong \sigma_x\ \overline{\overline{[z_q]}} \end{array}}{\Gamma, P, A \vdash_E f\ (\,\overline{e_n}\,) : \sigma_y'\ \overline{\overline{[z_r']}}}\ \textsc{Fun}$$

Règles de typage des appels de nœuds

$$\frac{\begin{array}{c} P \vdash f : \forall\, \overline{d_n}\,, \forall\, \overline{s_m}\,, \overline{typc_j} \Rightarrow \mathcal{T}_1 \to \mathcal{T}_2 \\ \Gamma, P, A \vdash_E (\,\overline{e_n}\,) : \mathcal{T}_1' \\ A \vdash \overline{typc_j[\,\overline{d_n \leftarrow d_n'}\;;\;\overline{s_m \leftarrow s_m'}\,]} \\ \mathcal{T}_1[\,\overline{d_n \leftarrow d_n'}\;;\;\overline{s_m \leftarrow s_m'}\,] = \sigma_x \quad \overline{\overline{[z_q]}} \\ \mathcal{T}_2[\,\overline{d_n \leftarrow d_n'}\;;\;\overline{s_m \leftarrow s_m'}\,] = \sigma_y' \quad \overline{\overline{[z_r']}} \\ \mathcal{T}_1' \cong \sigma_x \quad \overline{\overline{[\ell_h']}}\;\overline{\overline{[z_q]}} \end{array}}{\Gamma, P, A \vdash_E \quad f[\,\overline{\ell_h'}\,](\,\overline{e_n}\,) : \sigma_y' \quad \overline{\overline{[\ell_h']}}\;\overline{\overline{[z_r']}}}\ \textsc{Fun}$$

Règles de typage des appels de nœuds

$$\frac{\begin{array}{c} P \vdash f : \forall\, \overline{d_n}\,, \forall\, \overline{s_m}\,, \overline{typc_j} \Rightarrow \mathcal{T}_1 \to \mathcal{T}_2 \\ \Gamma, P, A \vdash_E (\,\overline{e_n}\,) : \mathcal{T}_1' \\ A \vdash \overline{typc_j[\,\overline{d_n \leftarrow d_n'}\;;\;\overline{s_m \leftarrow s_m'}\,]} \\ \mathcal{T}_1[\,\overline{d_n \leftarrow d_n'}\;;\;\overline{s_m \leftarrow s_m'}\,] = \overline{\sigma_x\,\overline{[\ell_g]}\,\overline{[z_q]}} \\ \mathcal{T}_2[\,\overline{d_n \leftarrow d_n'}\;;\;\overline{s_m \leftarrow s_m'}\,] = \overline{\sigma_y'\,\overline{[\ell_g]}\,\overline{[z_r']}} \\ \mathcal{T}_1' \cong \overline{\sigma_x\,\overline{[\ell_g]}\,\overline{[\ell_h']}\,\overline{[z_q]}} \end{array}}{\Gamma, P, A \vdash_E [\,\overline{\ell_g}\,] f [\,\overline{\ell_h'}\,](\,\overline{e_n}\,) : \overline{\sigma_y'\,\overline{[\ell_g]}\,\overline{[\ell_h']}\,\overline{[z_r']}}}\ \textsc{Fun}$$

Deux nouvelles constructions

```
node camellia (plaintext:b128, kw:b64[4], k:b64[18], ke:b64[4])
    returns (ciphertext:b128)
vars
   D1 : b64[12],
   D2 : b64[12]
let
  (D1[0],D2[0]) = plaintext ^ kw[0,1];
    ...
tel
```

Deux nouvelles constructions

```
node camellia (plaintext:b128, kw:b64[4], k:b64[18], ke:b64[4])
     returns (ciphertext:b128)
vars
    D1 : b64[12],
    D2 : b64[12]
let
  (D1[0],D2[0]) = plaintext ^ kw[0,1];
     ...
tel
```

- Les coercions explicites
- Les constructeurs de tableaux

Sémantique par évaluation

$$eval\_expr : architecture \rightarrow prog\_ctxt \rightarrow context \rightarrow expr \rightarrow option(list\ value)$$

1. On évalue tout dans l'ordre
2. Permet de gérer des équations de modifications
3. Sémantique la plus proche de celle implémenté

408 lignes

Sémantique relationnelle

$$eval\_expr\_to : architecture \rightarrow prog\_ctxt \rightarrow context \rightarrow expr \rightarrow list\ value \rightarrow Prop$$

1. Sémantique indépendante de l'ordre des équations
2. Sémantique non calculatoire
3. Accepte beaucoup de systèmes tel que $\{x = x \times 0\}$

265 lignes

Sémantique par tri topologique

$$\begin{aligned} &\mathit{eval\_expr}\ (\mathit{arch} : \mathit{architecture})\ (\mathit{prog} : \mathit{prog\_ctxt})\ (\mathit{eqns} : \mathit{list\ equations}) \\ &\qquad (\mathit{tctxt} : \mathit{typ\_ctxt})\ (\mathit{args} : \mathit{arguments})\ (e : \mathit{expr}) \\ &\qquad (a : \mathit{acc\_pred\ tctxt\ eqns\ e}) : \mathit{option\ value} \end{aligned}$$

1. Remonte l'ordre des évaluations pour calculer les valeurs
2. Peu maniable pour de la preuve de préservation de la sémantique
3. Contient de nombreux outils nécessaire pour un typeur

11602 lignes

Sémantique par point fixe

$$eval\_expr : architecture \rightarrow prog\_ctxt \rightarrow context \rightarrow expr \rightarrow option\ (Sum\ (list\ value)\ expr)$$

1. Essaye de calculer les équations par passages successifs sur le système
2. Sous ensemble stricte de la sémantique relationnelle

650 lignes

Conclusion

Il y a de nombreux problèmes en USUBA$_1$
On a vu plusieurs améliorations possible pour ces problèmes
Il existe d'autres améliorations comme l'élaboration de types et les boucles temporelles.

Merci de m'avoir écouté