Projet PCSN EI2018
Modélisation VHDL de l'algorithme de chiffrement AES

Jean-Max Dutertre, Olivier Potin, Guillaume Reymond, Jean-Baptiste Rigaud

MINES Saint-Étienne

Mercredi 4 décembre 2019

Objectifs des séances

1. Séance 1
   - S-box
   - SubBytes
2. Séance 2
3. Séance 3
4. Séance 4

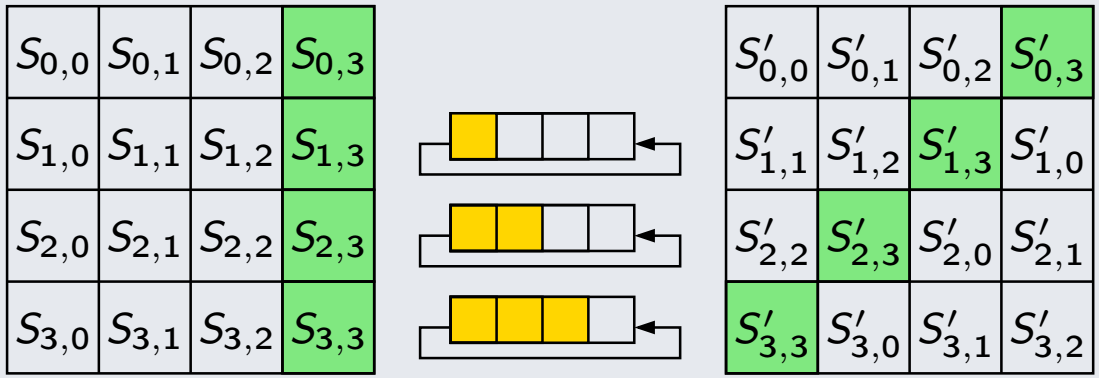
Objectifs des séances

1. Séance 1
2. Séance 2
   - ShiftRows
   - AddRoundKey
3. Séance 3
4. Séance 4

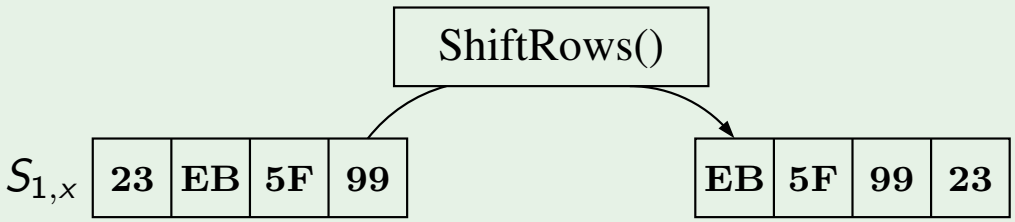
ShiftRows

Permutation cyclique (ou rotation)

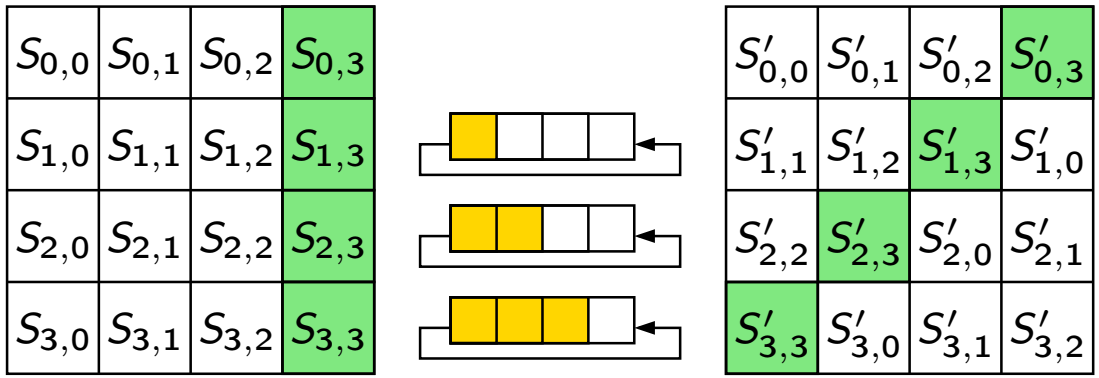
- Permute les octets de chaque ligne de l'état (State)
- Le décalage dépend de l'indice (0...3) de la ligne



Exemple



ShiftRows



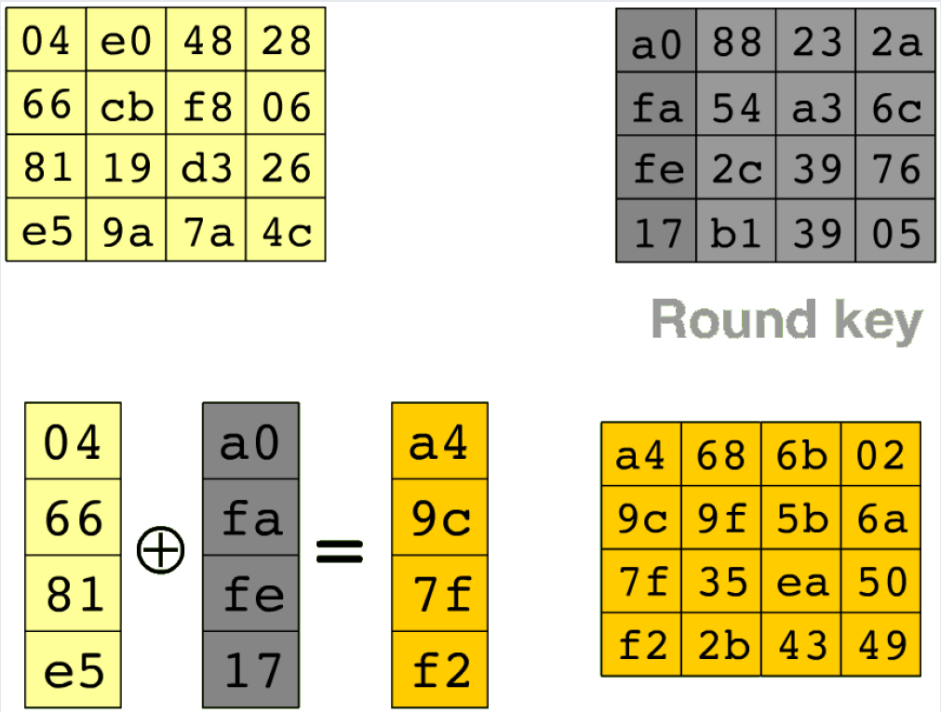
Implémentation

- Aucune fonction logique n'est appliquée
- Nécessite uniquement des fils pour assurer le décalage de chaque ligne

AddRoundKey

Ajout de la clé de ronde

- Ajoute la clé de ronde courante (ronde 0...10) à l'état courant
- L'addition dans $GF(2^8)$ est un OU-exclusif (XOR)
- Le XOR est appliqué bit à bit pour chaque octet

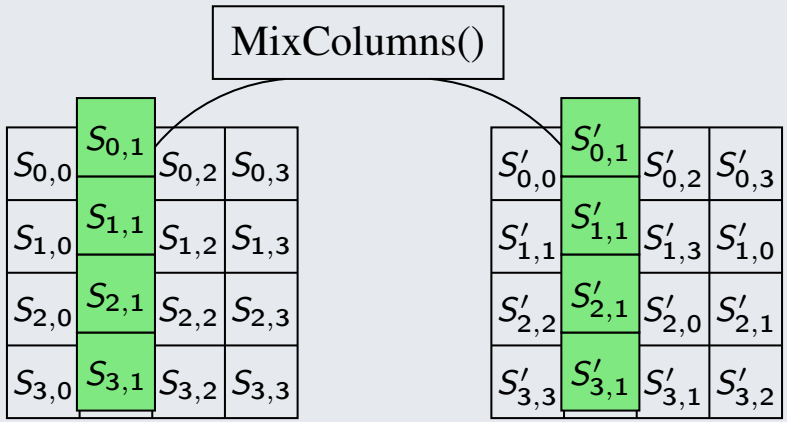


Objectifs des séances

1. Séance 1
2. Séance 2
3. Séance 3
   - MixColumns
   - Round Execution
4. Séance 4

MixColumns

Transformation linéaire colonne par colonne



Multiplication matricielle

$$\begin{bmatrix} S'_{0,c} \\ S'_{1,c} \\ S'_{2,c} \\ S'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} S_{0,c} \\ S_{1,c} \\ S_{2,c} \\ S_{3,c} \end{bmatrix}$$

MixColumns

Multiplication dans $GF(2^8)$

- Chaque élément est traité comme un polynôme dans $GF(2^8)$
- Multiplication notée $\otimes$, modulo le polynôme irréductible $x^8 + x^4 + x^3 + x + 1$, ou $b'100011011$

Multiplication par 02

- La valeur de l'octet est décalé à gauche (ex. `0xD4`) : $b'11010100 \ll 1 = b'11010100$
- Si le bit de poids fort vaut '1', réduction en utilisant un OU-exclusif avec $b'100011011$: $b'11010100 \oplus b'100011011 = b'010110011 = \texttt{0xB3}$

Multiplication par 03 : on notera que $03 = 02 + 01$

On pourra écrire : $\{03\} \otimes S_{r,c} = (\{02\} \otimes S_{r,c}) \oplus S_{r,c}$

MixColumns

Multiplication matricielle

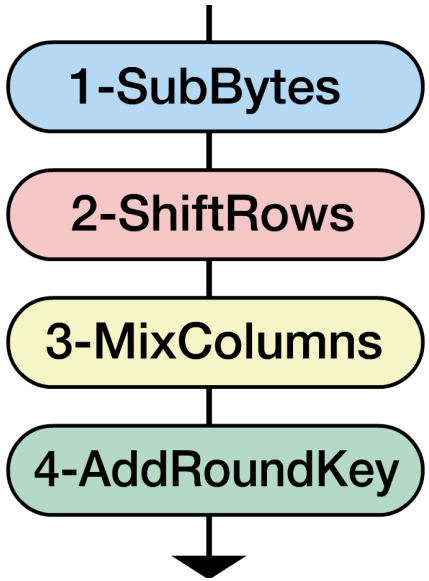
$$\begin{bmatrix} S'_{0,c} \\ S'_{1,c} \\ S'_{2,c} \\ S'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} S_{0,c} \\ S_{1,c} \\ S_{2,c} \\ S_{3,c} \end{bmatrix}$$

Ce qui est équivalent à :

$$\begin{aligned} S'_{0,c} &= (\{02\} \otimes S_{0,c}) \oplus (\{03\} \otimes S_{1,c}) \oplus S_{2,c} \oplus S_{3,c} \\ S'_{1,c} &= S_{0,c} \oplus (\{02\} \otimes S_{1,c}) \oplus (\{03\} \otimes S_{2,c}) \oplus S_{3,c} \\ S'_{2,c} &= S_{0,c} \oplus S_{1,c} \oplus (\{02\} \otimes S_{2,c}) \oplus (\{03\} \otimes S_{3,c}) \\ S'_{3,c} &= (\{03\} \otimes S_{0,c}) \oplus S_{1,c} \oplus S_{2,c} \oplus (\{02\} \otimes S_{0,c}) \end{aligned}$$

Round Execution

- Une ronde est constitué des 4 opérations SubBytes, ShiftRows, MixColumns et AddRoundKey
- Sauf pour la dernière ronde sans MixColumns
- L'entité MixColumns doit donc prévoir une entrée *enable*



Objectifs des séances

1. Séance 1
2. Séance 2
3. Séance 3
4. Séance 4
   - Registre de l'état courant *State*
   - Compteur
   - Machine d'états finis

Registre de l'état courant *State*

Pour chaque ronde, il est nécessaire de mémoriser l'état courant dans un registre.

Modification de RoundExecution

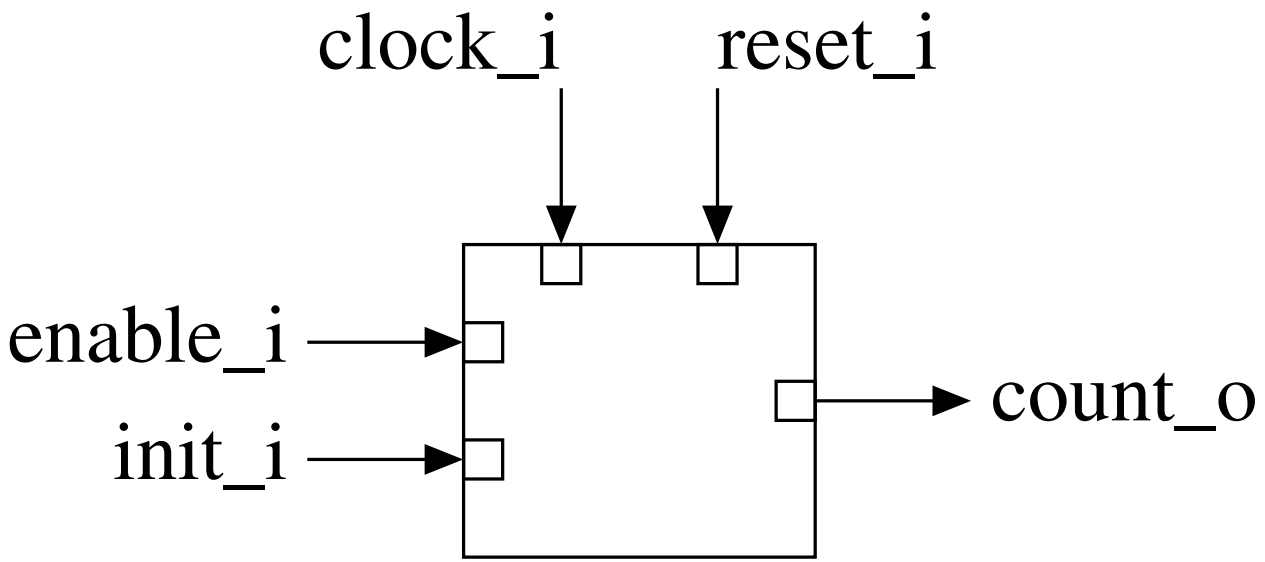
- Ajout d'un registre après l'instance AddRoundKey
- Ajout de l'horloge et du reset

```
signal state_s : bit128;
...
seq_0 : process (clock_i, reset_i) is
begin -- process seq_0
   if reset_i = '0' then -- asynchronous reset (active-low)
      state_s <= (others <= '0');
   elsif clock_i'event and clock_i = '1' then -- rising clock
      state_s <= AddRoundKey_output_s;
   end if;
end process seq_0;
```

Compteur

Utilisation du compteur de ronde

- Si le compteur est activé (enable = 1)
  - Si init = 1, le compteur vaut 0
  - Sinon, le compteur est incrémenté
- Sinon le compteur conserve sa valeur



Modélisation par un process VHDL

```
signal counter_s : bit4;
...
seq_0 : process (clock_i, reset_i, enable_i, init_i) is
begin -- process seq_0
   if reset_i = '0' then -- asynchronous reset (active-low)
      counter_s <= X"0";
   elsif clock_i'event and clock_i = '1' then -- rising clock
      if (enable_i = '1') then
         if (init_i = '1') then
            counter_s <= X"0";
         else
            counter_s <= counter_s + 1;
         end if;
      end if;
   else
      counter_s <= counter_s;
   end if;
end process seq_0;
```

Machine d'états finis

Rôle de la machine d'état

La machine d'état finis gère l'exécution de l'algorithme AES

- Lancement du chiffrement lorsque `start_i` = 1
- Sélection de la clé de ronde
- Gestion du compteur de ronde
- Fin de calcul : `aes_on_o` remis à 1

Elle pilote les blocs *Round Execution* et *Key Expansion*

Key expansion

Fonctionnement

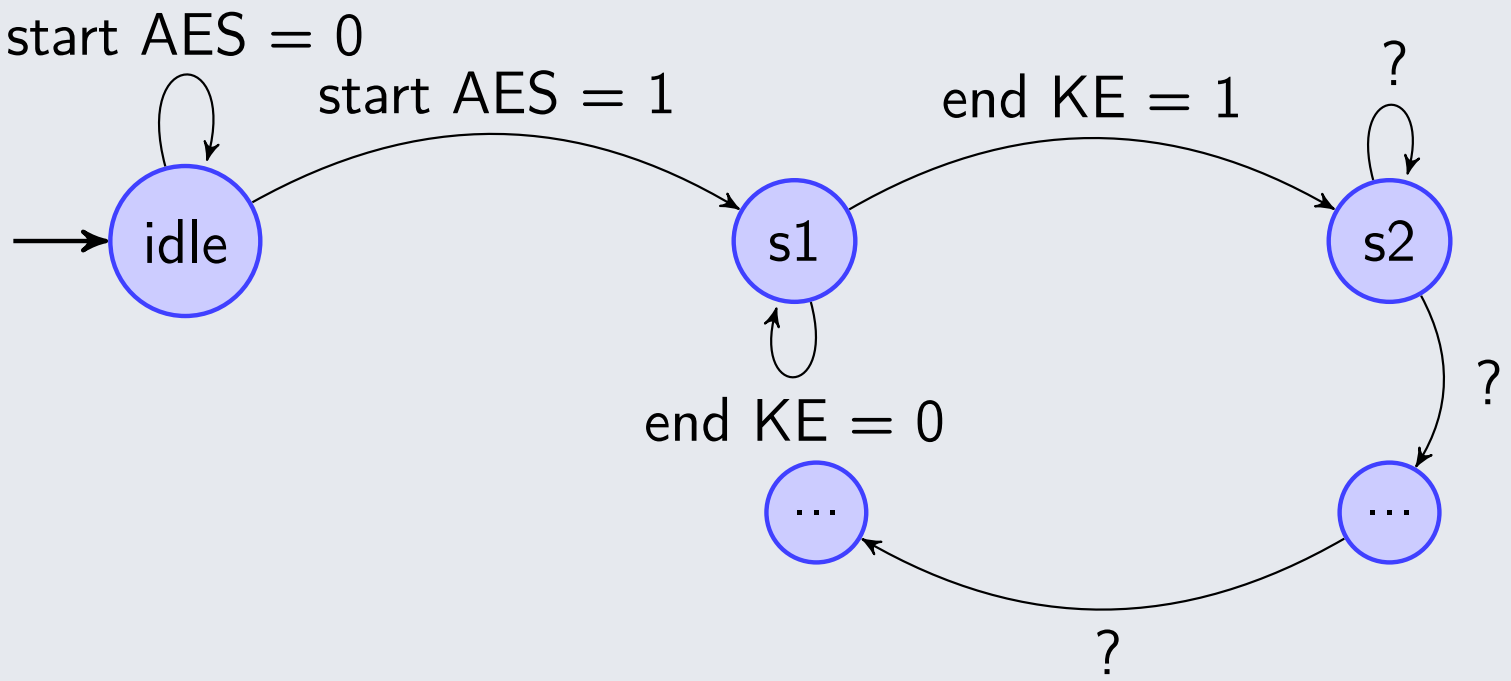
L'entité key expansion fournit une clé de ronde `expansion_key_o` en fonction de `round_i`

- Entrées : clé `key_i`, `start_i`, `round_i`
- L'ensemble des clés de ronde sont prêtes lorsque `end_o` $= 1$
- Sorties : sous-clé `expansion_key_o` de la ronde `round_i`

Il faut donc attendre `end_o` $= 1$ (fin de l'expansion de clé) avant de lancer l'exécution des rondes

Machine d'états finis

Le (début du) diagramme de la machine d'état



Avec s1 : start key expansion et s2 : start round execution