- Agenda: explore the **Advanced Encryption Standard (AES)**, i.e.,

$$\textbf{Square}\ [4] \rightsquigarrow \textbf{Rijndael}\ [5] \rightsquigarrow \textbf{AES}\ [2, 8],$$

  via

  1. an "in theory", i.e., design-oriented perspective, and
  2. an "in practice", i.e., implementation-oriented perspective,

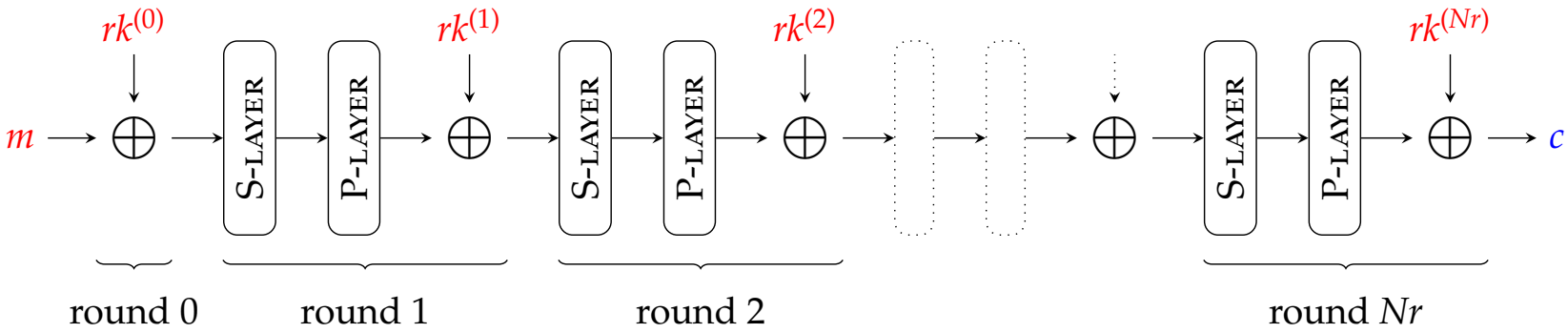
- Caveat!

$$\sim 2 \text{ hours} \quad \Rightarrow \quad \text{introductory, and (very) selective (versus definitive) coverage.}$$

- AES is an (iterated) block cipher, where

$$\begin{aligned} \textsc{Enc} &: \{0,1\}^{8\cdot 4\cdot Nk} \times \{0,1\}^{8\cdot 4\cdot Nb} \rightarrow \{0,1\}^{8\cdot 4\cdot Nb} \\ \textsc{Dec} &: \{0,1\}^{8\cdot 4\cdot Nk} \times \{0,1\}^{8\cdot 4\cdot Nb} \rightarrow \{0,1\}^{8\cdot 4\cdot Nb} \end{aligned}$$

are realised by using a **substitution-permutation network**



i.e., $Nr + 1$ **rounds**: each r-th round

- applies one or more **round functions**,
- involves a **round key** $rk^{(r)}$ derived from the **cipher key** k.

- AES is actually a *family* of block ciphers: per [8, Figure 3], the parameter sets are

	Nk	*Nb*	*Nr*
AES-128	4	4	10
AES-192	6	4	12
AES-256	8	4	14

but we'll focus *exclusively* on AES-128 encryption only.

- AES [8, Section 4] operates on elements in the **finite field** $\mathbb{F}_{2^8}$, which is realised concretely as

$$\mathbb{F}_2[\mathbf{x}]/p(\mathbf{x})$$

where

$$p(\mathbf{x}) = \mathbf{x}^8 + \mathbf{x}^4 + \mathbf{x}^3 + \mathbf{x} + 1$$

meaning

- a given field element is represented using a polynomial whose **indeterminate** is $\mathbf{x}$,
- coefficients of such polynomials are in the field $\mathbb{F}_2$, and
- arithmetic with field elements is modulo an **irreducible polynomial** $p(\mathbf{x}) = \mathbf{x}^8 + \mathbf{x}^4 + \mathbf{x}^3 + \mathbf{x} + 1$.

- Beware!
  - a hexadecimal short-hand is used for immediate field elements, e.g.,

$$13 \mapsto 13_{(16)} \equiv 00010011_{(2)} \equiv \langle 1,1,0,0,1,0,0,0 \rangle_{(\mathbf{x})} \mapsto \mathbf{x}^4 + \mathbf{x} + 1,$$

    and
  - to avoid confusion, we carefully highlight where arithmetic in some field F is required, e.g.,

$$\begin{aligned} \oplus_F &\mapsto \text{“addition in the field } F\text{”} \\ \ominus_F &\mapsto \text{“subtraction in the field } F\text{”} \\ \otimes_F &\mapsto \text{“multiplication in the field } F\text{”} \\ \oslash_F &\mapsto \text{“division in the field } F\text{”} \end{aligned}$$

- Beware!

- we can ignore/avoid *most* field arithmetic, but rely on at least:
  1. **addition**, i.e.,
  $$r = x \oplus_{\mathbb{F}_{2^8}} y \mapsto x \oplus y.$$
  2. **multiplication-by-x**, i.e.,
  $$r = \texttt{xtimes}(x) = \mathbf{x} \otimes_{\mathbb{F}_{2^8}} x \mapsto \begin{cases} \langle 0, x_0, x_1, x_2, x_3, x_4, x_5, x_6 \rangle \oplus \langle 1, 1, 0, 1, 1, 0, 0, 0 \rangle & \text{if } x_7 = 1 \\ \langle 0, x_0, x_1, x_2, x_3, x_4, x_5, x_6 \rangle & \text{otherwise} \end{cases}$$
  3. **multiplication-by-c**, e.g.,
  $$\begin{aligned} r &= \texttt{01} \otimes_{\mathbb{F}_{2^8}} x = 1 \otimes_{\mathbb{F}_{2^8}} x = x \\ r &= \texttt{02} \otimes_{\mathbb{F}_{2^8}} x = \mathbf{x} \otimes_{\mathbb{F}_{2^8}} x = \texttt{xtimes}(x) \\ r &= \texttt{03} \otimes_{\mathbb{F}_{2^8}} x = (\mathbf{x} + 1) \otimes_{\mathbb{F}_{2^8}} x = \texttt{xtimes}(x) \oplus x \end{aligned}$$

- Beware!

- we can ignore/avoid *most* field arithmetic, but rely on at least:
  1. **addition**, i.e.,
  $$r = x \oplus_{\mathbb{F}_{2^8}} y \mapsto x \oplus y.$$
  2. **multiplication-by-x**, i.e.,
  $$r = \texttt{xtimes}(x) = \texttt{x} \otimes_{\mathbb{F}_{2^8}} x \mapsto \begin{cases} (x \ll 1) \oplus \texttt{11B} & \text{if } x_7 = 1 \\ x \ll 1 & \text{otherwise} \end{cases}$$
  3. **multiplication-by-c**, e.g.,
  $$\begin{aligned} r &= \texttt{01} \otimes_{\mathbb{F}_{2^8}} x = 1 \otimes_{\mathbb{F}_{2^8}} x = x \\ r &= \texttt{02} \otimes_{\mathbb{F}_{2^8}} x = \texttt{x} \otimes_{\mathbb{F}_{2^8}} x = \texttt{xtimes}(x) \\ r &= \texttt{03} \otimes_{\mathbb{F}_{2^8}} x = (\texttt{x} + 1) \otimes_{\mathbb{F}_{2^8}} x = \texttt{xtimes}(x) \oplus x \end{aligned}$$

- AES [8, Section 3.4] organises field elements into (4×4)-element matrices, e.g.,
  1. the r-th **state matrix**

$$s^{(r)} = \begin{bmatrix} s^{(r)}_{0,0} & s^{(r)}_{0,1} & s^{(r)}_{0,2} & s^{(r)}_{0,3} \\ s^{(r)}_{1,0} & s^{(r)}_{1,1} & s^{(r)}_{1,2} & s^{(r)}_{1,3} \\ s^{(r)}_{2,0} & s^{(r)}_{2,1} & s^{(r)}_{2,2} & s^{(r)}_{2,3} \\ s^{(r)}_{3,0} & s^{(r)}_{3,1} & s^{(r)}_{3,2} & s^{(r)}_{3,3} \end{bmatrix}$$

     for each $s^{(r)}_{i,j} \in \mathbb{F}_{2^8}$, or
  2. the r-th **round key matrix**

$$rk^{(r)} = \begin{bmatrix} rk^{(r)}_{0,0} & rk^{(r)}_{0,1} & rk^{(r)}_{0,2} & rk^{(r)}_{0,3} \\ rk^{(r)}_{1,0} & rk^{(r)}_{1,1} & rk^{(r)}_{1,2} & rk^{(r)}_{1,3} \\ rk^{(r)}_{2,0} & rk^{(r)}_{2,1} & rk^{(r)}_{2,2} & rk^{(r)}_{2,3} \\ rk^{(r)}_{3,0} & rk^{(r)}_{3,1} & rk^{(r)}_{3,2} & rk^{(r)}_{3,3} \end{bmatrix}$$

     for each $rk^{(r)}_{i,j} \in \mathbb{F}_{2^8}$,

  which can be read from (resp. written to) in column-wise order.

- AES [8, Section 5.1.1] uses a single S-box, defined as the composition of two functions, i.e.,

$$\text{S-box}(x) = (f \circ g)(x) = f(g(x)),$$

where

$$g(x) = \begin{cases} 0 & \text{if } x = 0 \\ 1 \oslash_{\mathbb{F}_{2^8}} x & \text{otherwise} \end{cases}$$

i.e., g is a field (pseudo-)inversion, and

$$f\left(\begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix}\right) = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \otimes_{\mathbb{F}_2} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} \oplus_{\mathbb{F}_2} \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

is an affine transformation.

Part 1: in theory (6)
Low(er)-level concepts/components

- AES [8, Section 5.1.1] uses a single S-box, defined as the composition of two functions, i.e.,

$$\text{S-BOX}(x) \;=\; (f \circ g)(x) \;=\; f(g(x)),$$

where

$$g(x) = \begin{cases} 0 & \text{if } x = 0 \\ 1 \oslash_{\mathbb{F}_{2^8}} x & \text{otherwise} \end{cases}$$

i.e., g is a field (pseudo-)inversion, and

$$f\left(\begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix}\right) = \begin{bmatrix} x_0 \oplus_{\mathbb{F}_2} x_4 \oplus_{\mathbb{F}_2} x_5 \oplus_{\mathbb{F}_2} x_6 \oplus_{\mathbb{F}_2} x_7 \oplus_{\mathbb{F}_2} 1 \\ x_0 \oplus_{\mathbb{F}_2} x_1 \oplus_{\mathbb{F}_2} x_5 \oplus_{\mathbb{F}_2} x_6 \oplus_{\mathbb{F}_2} x_7 \oplus_{\mathbb{F}_2} 1 \\ x_0 \oplus_{\mathbb{F}_2} x_1 \oplus_{\mathbb{F}_2} x_2 \oplus_{\mathbb{F}_2} x_6 \oplus_{\mathbb{F}_2} x_7 \oplus_{\mathbb{F}_2} 0 \\ x_0 \oplus_{\mathbb{F}_2} x_1 \oplus_{\mathbb{F}_2} x_2 \oplus_{\mathbb{F}_2} x_3 \oplus_{\mathbb{F}_2} x_7 \oplus_{\mathbb{F}_2} 0 \\ x_0 \oplus_{\mathbb{F}_2} x_1 \oplus_{\mathbb{F}_2} x_2 \oplus_{\mathbb{F}_2} x_3 \oplus_{\mathbb{F}_2} x_4 \oplus_{\mathbb{F}_2} 0 \\ x_1 \oplus_{\mathbb{F}_2} x_2 \oplus_{\mathbb{F}_2} x_3 \oplus_{\mathbb{F}_2} x_4 \oplus_{\mathbb{F}_2} x_5 \oplus_{\mathbb{F}_2} 1 \\ x_2 \oplus_{\mathbb{F}_2} x_3 \oplus_{\mathbb{F}_2} x_4 \oplus_{\mathbb{F}_2} x_5 \oplus_{\mathbb{F}_2} x_6 \oplus_{\mathbb{F}_2} 1 \\ x_3 \oplus_{\mathbb{F}_2} x_4 \oplus_{\mathbb{F}_2} x_5 \oplus_{\mathbb{F}_2} x_6 \oplus_{\mathbb{F}_2} x_7 \oplus_{\mathbb{F}_2} 0 \end{bmatrix}$$

is an affine transformation.

Algorithm (AES round functions [8, Section 5.1])

Input: A state matrix $s^{(r)}$, and a round key matrix $rk^{(r)}$
Output: A state matrix $s'^{(r)} = \texttt{AddRoundKey}(s^{(r)}, rk^{(r)})$

$$s'^{(r)} = \texttt{AddRoundKey}(s^{(r)}, rk^{(r)})$$

$$= \texttt{AddRoundKey}\left(\begin{bmatrix} s^{(r)}_{0,0} & s^{(r)}_{0,1} & s^{(r)}_{0,2} & s^{(r)}_{0,3} \\ s^{(r)}_{1,0} & s^{(r)}_{1,1} & s^{(r)}_{1,2} & s^{(r)}_{1,3} \\ s^{(r)}_{2,0} & s^{(r)}_{2,1} & s^{(r)}_{2,2} & s^{(r)}_{2,3} \\ s^{(r)}_{3,0} & s^{(r)}_{3,1} & s^{(r)}_{3,2} & s^{(r)}_{3,3} \end{bmatrix}, \begin{bmatrix} rk^{(r)}_{0,0} & rk^{(r)}_{0,1} & rk^{(r)}_{0,2} & rk^{(r)}_{0,3} \\ rk^{(r)}_{1,0} & rk^{(r)}_{1,1} & rk^{(r)}_{1,2} & rk^{(r)}_{1,3} \\ rk^{(r)}_{2,0} & rk^{(r)}_{2,1} & rk^{(r)}_{2,2} & rk^{(r)}_{2,3} \\ rk^{(r)}_{3,0} & rk^{(r)}_{3,1} & rk^{(r)}_{3,2} & rk^{(r)}_{3,3} \end{bmatrix}\right)$$

$$= \begin{bmatrix} \left(s^{(r)}_{0,0} \oplus_{\mathbb{F}_{2^8}} rk^{(r)}_{0,0}\right) & \left(s^{(r)}_{0,1} \oplus_{\mathbb{F}_{2^8}} rk^{(r)}_{0,1}\right) & \left(s^{(r)}_{0,2} \oplus_{\mathbb{F}_{2^8}} rk^{(r)}_{0,2}\right) & \left(s^{(r)}_{0,3} \oplus_{\mathbb{F}_{2^8}} rk^{(r)}_{0,3}\right) \\ \left(s^{(r)}_{1,0} \oplus_{\mathbb{F}_{2^8}} rk^{(r)}_{1,0}\right) & \left(s^{(r)}_{1,1} \oplus_{\mathbb{F}_{2^8}} rk^{(r)}_{1,1}\right) & \left(s^{(r)}_{1,2} \oplus_{\mathbb{F}_{2^8}} rk^{(r)}_{1,2}\right) & \left(s^{(r)}_{1,3} \oplus_{\mathbb{F}_{2^8}} rk^{(r)}_{1,3}\right) \\ \left(s^{(r)}_{2,0} \oplus_{\mathbb{F}_{2^8}} rk^{(r)}_{2,0}\right) & \left(s^{(r)}_{2,1} \oplus_{\mathbb{F}_{2^8}} rk^{(r)}_{2,1}\right) & \left(s^{(r)}_{2,2} \oplus_{\mathbb{F}_{2^8}} rk^{(r)}_{2,2}\right) & \left(s^{(r)}_{2,3} \oplus_{\mathbb{F}_{2^8}} rk^{(r)}_{2,3}\right) \\ \left(s^{(r)}_{3,0} \oplus_{\mathbb{F}_{2^8}} rk^{(r)}_{3,0}\right) & \left(s^{(r)}_{3,1} \oplus_{\mathbb{F}_{2^8}} rk^{(r)}_{3,1}\right) & \left(s^{(r)}_{3,2} \oplus_{\mathbb{F}_{2^8}} rk^{(r)}_{3,2}\right) & \left(s^{(r)}_{3,3} \oplus_{\mathbb{F}_{2^8}} rk^{(r)}_{3,3}\right) \end{bmatrix}$$

Algorithm (AES round functions [8, Section 5.1])

Input: A state matrix $s^{(r)}$
Output: A state matrix $s'^{(r)} = \texttt{SubBytes}(s^{(r)})$

$$
\begin{aligned}
s'^{(r)} &= \texttt{SubBytes}(s^{(r)}) \\
&= \texttt{SubBytes}\left(\begin{bmatrix} s^{(r)}_{0,0} & s^{(r)}_{0,1} & s^{(r)}_{0,2} & s^{(r)}_{0,3} \\ s^{(r)}_{1,0} & s^{(r)}_{1,1} & s^{(r)}_{1,2} & s^{(r)}_{1,3} \\ s^{(r)}_{2,0} & s^{(r)}_{2,1} & s^{(r)}_{2,2} & s^{(r)}_{2,3} \\ s^{(r)}_{3,0} & s^{(r)}_{3,1} & s^{(r)}_{3,2} & s^{(r)}_{3,3} \end{bmatrix}\right) \\
&= \begin{bmatrix} \text{S-BOX}\left(s^{(r)}_{0,0}\right) & \text{S-BOX}\left(s^{(r)}_{0,1}\right) & \text{S-BOX}\left(s^{(r)}_{0,2}\right) & \text{S-BOX}\left(s^{(r)}_{0,3}\right) \\ \text{S-BOX}\left(s^{(r)}_{1,0}\right) & \text{S-BOX}\left(s^{(r)}_{1,1}\right) & \text{S-BOX}\left(s^{(r)}_{1,2}\right) & \text{S-BOX}\left(s^{(r)}_{1,3}\right) \\ \text{S-BOX}\left(s^{(r)}_{2,0}\right) & \text{S-BOX}\left(s^{(r)}_{2,1}\right) & \text{S-BOX}\left(s^{(r)}_{2,2}\right) & \text{S-BOX}\left(s^{(r)}_{2,3}\right) \\ \text{S-BOX}\left(s^{(r)}_{3,0}\right) & \text{S-BOX}\left(s^{(r)}_{3,1}\right) & \text{S-BOX}\left(s^{(r)}_{3,2}\right) & \text{S-BOX}\left(s^{(r)}_{3,3}\right) \end{bmatrix}
\end{aligned}
$$

Algorithm (AES round functions [8, Section 5.1])

Input: A state matrix $s^{(r)}$
Output: A state matrix $s'^{(r)} = \texttt{ShiftRows}(s^{(r)})$

$$
\begin{aligned}
s'^{(r)} &= \texttt{ShiftRows}(s^{(r)}) \\
&= \texttt{ShiftRows}\left(\begin{bmatrix} s^{(r)}_{0,0} & s^{(r)}_{0,1} & s^{(r)}_{0,2} & s^{(r)}_{0,3} \\ s^{(r)}_{1,0} & s^{(r)}_{1,1} & s^{(r)}_{1,2} & s^{(r)}_{1,3} \\ s^{(r)}_{2,0} & s^{(r)}_{2,1} & s^{(r)}_{2,2} & s^{(r)}_{2,3} \\ s^{(r)}_{3,0} & s^{(r)}_{3,1} & s^{(r)}_{3,2} & s^{(r)}_{3,3} \end{bmatrix}\right) \\
&= \begin{bmatrix} s^{(r)}_{0,0} & s^{(r)}_{0,1} & s^{(r)}_{0,2} & s^{(r)}_{0,3} \\ s^{(r)}_{1,1} & s^{(r)}_{1,2} & s^{(r)}_{1,3} & s^{(r)}_{1,0} \\ s^{(r)}_{2,2} & s^{(r)}_{2,3} & s^{(r)}_{2,0} & s^{(r)}_{2,1} \\ s^{(r)}_{3,3} & s^{(r)}_{3,0} & s^{(r)}_{3,1} & s^{(r)}_{3,2} \end{bmatrix}
\end{aligned}
$$

Algorithm (AES round functions [8, Section 5.1])

Input: A state matrix $s^{(r)}$
Output: A state matrix $s'^{(r)} = \texttt{MixColumns}(s^{(r)})$

$$s'^{(r)} = \texttt{MixColumns}(s^{(r)})$$

$$= \texttt{MixColumns}\left(\begin{bmatrix} s^{(r)}_{0,0} & s^{(r)}_{0,1} & s^{(r)}_{0,2} & s^{(r)}_{0,3} \\ s^{(r)}_{1,0} & s^{(r)}_{1,1} & s^{(r)}_{1,2} & s^{(r)}_{1,3} \\ s^{(r)}_{2,0} & s^{(r)}_{2,1} & s^{(r)}_{2,2} & s^{(r)}_{2,3} \\ s^{(r)}_{3,0} & s^{(r)}_{3,1} & s^{(r)}_{3,2} & s^{(r)}_{3,3} \end{bmatrix}\right)$$

where each column of the result, i.e., for each $0 \leq j < 4$, is produced via

$$\begin{bmatrix} s'^{(r)}_{0,j} \\ s'^{(r)}_{1,j} \\ s'^{(r)}_{2,j} \\ s'^{(r)}_{3,j} \end{bmatrix} = \texttt{MixColumn}\left(\begin{bmatrix} s^{(r)}_{0,j} \\ s^{(r)}_{1,j} \\ s^{(r)}_{2,j} \\ s^{(r)}_{3,j} \end{bmatrix}\right) = \begin{bmatrix} \texttt{02} & \texttt{03} & \texttt{01} & \texttt{01} \\ \texttt{01} & \texttt{02} & \texttt{03} & \texttt{01} \\ \texttt{01} & \texttt{01} & \texttt{02} & \texttt{03} \\ \texttt{03} & \texttt{01} & \texttt{01} & \texttt{02} \end{bmatrix} \otimes_{\mathbb{F}_{2^8}} \begin{bmatrix} s^{(r)}_{0,j} \\ s^{(r)}_{1,j} \\ s^{(r)}_{2,j} \\ s^{(r)}_{3,j} \end{bmatrix}$$

$$= \begin{bmatrix} \left(\texttt{02} \otimes_{\mathbb{F}_{2^8}} s^{(r)}_{0,j}\right) \oplus_{\mathbb{F}_{2^8}} \left(\texttt{03} \otimes_{\mathbb{F}_{2^8}} s^{(r)}_{1,j}\right) \oplus_{\mathbb{F}_{2^8}} \left(\texttt{01} \otimes_{\mathbb{F}_{2^8}} s^{(r)}_{2,j}\right) \oplus_{\mathbb{F}_{2^8}} \left(\texttt{01} \otimes_{\mathbb{F}_{2^8}} s^{(r)}_{3,j}\right) \\ \left(\texttt{01} \otimes_{\mathbb{F}_{2^8}} s^{(r)}_{0,j}\right) \oplus_{\mathbb{F}_{2^8}} \left(\texttt{02} \otimes_{\mathbb{F}_{2^8}} s^{(r)}_{1,j}\right) \oplus_{\mathbb{F}_{2^8}} \left(\texttt{03} \otimes_{\mathbb{F}_{2^8}} s^{(r)}_{2,j}\right) \oplus_{\mathbb{F}_{2^8}} \left(\texttt{01} \otimes_{\mathbb{F}_{2^8}} s^{(r)}_{3,j}\right) \\ \left(\texttt{01} \otimes_{\mathbb{F}_{2^8}} s^{(r)}_{0,j}\right) \oplus_{\mathbb{F}_{2^8}} \left(\texttt{01} \otimes_{\mathbb{F}_{2^8}} s^{(r)}_{1,j}\right) \oplus_{\mathbb{F}_{2^8}} \left(\texttt{02} \otimes_{\mathbb{F}_{2^8}} s^{(r)}_{2,j}\right) \oplus_{\mathbb{F}_{2^8}} \left(\texttt{03} \otimes_{\mathbb{F}_{2^8}} s^{(r)}_{3,j}\right) \\ \left(\texttt{03} \otimes_{\mathbb{F}_{2^8}} s^{(r)}_{0,j}\right) \oplus_{\mathbb{F}_{2^8}} \left(\texttt{01} \otimes_{\mathbb{F}_{2^8}} s^{(r)}_{1,j}\right) \oplus_{\mathbb{F}_{2^8}} \left(\texttt{01} \otimes_{\mathbb{F}_{2^8}} s^{(r)}_{2,j}\right) \oplus_{\mathbb{F}_{2^8}} \left(\texttt{02} \otimes_{\mathbb{F}_{2^8}} s^{(r)}_{3,j}\right) \end{bmatrix}$$

Algorithm (AES-128.Enc-KeyExpand [8, Section 5.2])

Input: A 128-bit cipher key k
Output: An 11-element sequence $rk = \text{AES-128.Enc-KeyExpand}(k) \simeq \texttt{ExpandRoundKey}(k)$ of round keys

Generate a sequence of column vectors

$$\overbrace{\begin{bmatrix} w_{0,0} \\ w_{1,0} \\ w_{2,0} \\ w_{3,0} \end{bmatrix}\begin{bmatrix} w_{0,1} \\ w_{1,1} \\ w_{2,1} \\ w_{3,1} \end{bmatrix}\begin{bmatrix} w_{0,2} \\ w_{1,2} \\ w_{2,2} \\ w_{3,2} \end{bmatrix}\begin{bmatrix} w_{0,3} \\ w_{1,3} \\ w_{2,3} \\ w_{3,3} \end{bmatrix}}^{\text{round key } rk^{(0)}}\overbrace{\begin{bmatrix} w_{0,4} \\ w_{1,4} \\ w_{2,4} \\ w_{3,4} \end{bmatrix}\begin{bmatrix} w_{0,5} \\ w_{1,5} \\ w_{2,5} \\ w_{3,5} \end{bmatrix}\begin{bmatrix} w_{0,6} \\ w_{1,6} \\ w_{2,6} \\ w_{3,6} \end{bmatrix}\begin{bmatrix} w_{0,7} \\ w_{1,7} \\ w_{2,7} \\ w_{3,7} \end{bmatrix}}^{\text{round key } rk^{(1)}} \ldots$$

using the following rules

1. in the j-th column-vector for $0 \leq j < Nk$, we set $w_{i,j} = k_{i,j}$ to match the cipher key,
2. in the j-th column-vector for $Nk \leq j < Nb \cdot (Nr + 1)$ we set

$$\begin{bmatrix} w_{0,j} \\ w_{1,j} \\ w_{2,j} \\ w_{3,j} \end{bmatrix} = \begin{cases} \begin{bmatrix} rc_{j/Nk} \oplus_{\mathbb{F}_{2^8}} \text{S-box}(w_{1,j-1}) \oplus_{\mathbb{F}_{2^8}} w_{0,j-Nk} \\ \text{S-box}(w_{2,j-1}) \oplus_{\mathbb{F}_{2^8}} w_{1,j-Nk} \\ \text{S-box}(w_{3,j-1}) \oplus_{\mathbb{F}_{2^8}} w_{2,j-Nk} \\ \text{S-box}(w_{0,j-1}) \oplus_{\mathbb{F}_{2^8}} w_{3,j-Nk} \end{bmatrix} & \text{if } j = 0 \pmod{Nk} \\ \begin{bmatrix} w_{0,j-1} \oplus_{\mathbb{F}_{2^8}} w_{0,j-Nk} \\ w_{1,j-1} \oplus_{\mathbb{F}_{2^8}} w_{1,j-Nk} \\ w_{2,j-1} \oplus_{\mathbb{F}_{2^8}} w_{2,j-Nk} \\ w_{3,j-1} \oplus_{\mathbb{F}_{2^8}} w_{3,j-Nk} \end{bmatrix} & \text{otherwise} \end{cases}$$

then combining them to yield a sequence of round keys, where $rc^{(r)} = \mathbf{x}^{r-1}$ denotes the r-th **round constant**.

Algorithm (AES-128.Enc-KeyEvolve [8, Section 5.2])

Input: The r-th round key matrix $rk^{(r)}$ and round constant $rc^{(r)}$
Output: The $(r+1)$-th round key matrix $rk^{(r+1)} = \text{AES-128.Enc-KeyEvolve}(k) \simeq \text{“EvolveRoundKey”}(rk^{(r)}, rc^{(r)})$

1. Compute

$$\begin{aligned}
rk_{0,0}^{(r+1)} &\leftarrow rc^{(r)} \oplus_{\mathbb{F}_{2^8}} \text{S-box}\left(rk_{1,3}^{(r)}\right) \oplus_{\mathbb{F}_{2^8}} rk_{0,0}^{(r)} \\
rk_{1,0}^{(r+1)} &\leftarrow \text{S-box}\left(rk_{2,3}^{(r)}\right) \oplus_{\mathbb{F}_{2^8}} rk_{1,0}^{(r)} \\
rk_{2,0}^{(r+1)} &\leftarrow \text{S-box}\left(rk_{3,3}^{(r)}\right) \oplus_{\mathbb{F}_{2^8}} rk_{2,0}^{(r)} \\
rk_{3,0}^{(r+1)} &\leftarrow \text{S-box}\left(rk_{0,3}^{(r)}\right) \oplus_{\mathbb{F}_{2^8}} rk_{3,0}^{(r)}
\end{aligned}$$

2. Compute

$$\begin{aligned}
rk_{0,1}^{(r+1)} &\leftarrow rk_{0,0}^{(r+1)} \oplus_{\mathbb{F}_{2^8}} rk_{0,1}^{(r)} & rk_{1,1}^{(r+1)} &\leftarrow rk_{1,0}^{(r+1)} \oplus_{\mathbb{F}_{2^8}} rk_{1,1}^{(r)} \\
rk_{2,1}^{(r+1)} &\leftarrow rk_{2,0}^{(r+1)} \oplus_{\mathbb{F}_{2^8}} rk_{2,1}^{(r)} & rk_{3,1}^{(r+1)} &\leftarrow rk_{3,0}^{(r+1)} \oplus_{\mathbb{F}_{2^8}} rk_{3,1}^{(r)} \\
rk_{0,2}^{(r+1)} &\leftarrow rk_{0,1}^{(r+1)} \oplus_{\mathbb{F}_{2^8}} rk_{0,2}^{(r)} & rk_{1,2}^{(r+1)} &\leftarrow rk_{1,1}^{(r+1)} \oplus_{\mathbb{F}_{2^8}} rk_{1,2}^{(r)} \\
rk_{2,2}^{(r+1)} &\leftarrow rk_{2,1}^{(r+1)} \oplus_{\mathbb{F}_{2^8}} rk_{2,2}^{(r)} & rk_{3,2}^{(r+1)} &\leftarrow rk_{3,1}^{(r+1)} \oplus_{\mathbb{F}_{2^8}} rk_{3,2}^{(r)} \\
rk_{0,3}^{(r+1)} &\leftarrow rk_{0,2}^{(r+1)} \oplus_{\mathbb{F}_{2^8}} rk_{0,3}^{(r)} & rk_{1,3}^{(r+1)} &\leftarrow rk_{1,2}^{(r+1)} \oplus_{\mathbb{F}_{2^8}} rk_{1,3}^{(r)} \\
rk_{2,3}^{(r+1)} &\leftarrow rk_{2,2}^{(r+1)} \oplus_{\mathbb{F}_{2^8}} rk_{2,3}^{(r)} & rk_{3,3}^{(r+1)} &\leftarrow rk_{3,2}^{(r+1)} \oplus_{\mathbb{F}_{2^8}} rk_{3,3}^{(r)}
\end{aligned}$$

3. Return $rk^{(r+1)}$.

Algorithm (AES-128.Enc [8, Section 5.1])

Input: A 128-bit cipher key k, and a 128-bit plaintext message m
Output: A 128-bit ciphertext message $c = \text{AES-128.Enc}(k, m)$

$rk \leftarrow \texttt{ExpandRoundKey}(k)$
$s \leftarrow m$
$s \leftarrow \texttt{AddRoundKey}(s, rk^{(0)})$
for $r = 1$ **upto** $Nr - 1$ **step** $+1$ **do**
  $s \leftarrow \texttt{SubBytes}(s)$
  $s \leftarrow \texttt{ShiftRows}(s)$
  $s \leftarrow \texttt{MixColumns}(s)$
  $s \leftarrow \texttt{AddRoundKey}(s, rk^{(r)})$
end
$s \leftarrow \texttt{SubBytes}(s)$
$s \leftarrow \texttt{ShiftRows}(s)$
$s \leftarrow \texttt{AddRoundKey}(s, rk^{(10)})$
$c \leftarrow s$
return c

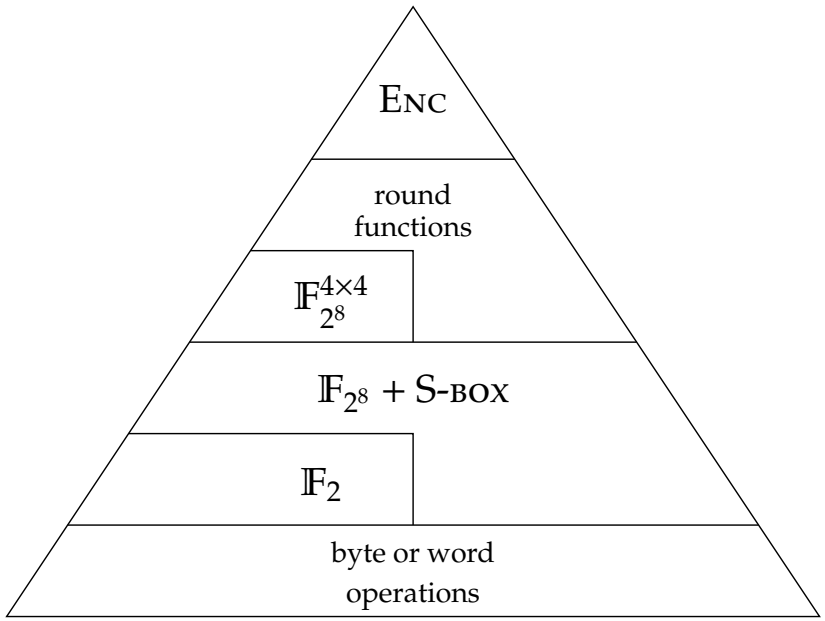
Algorithm (AES-128.Enc [8, Section 5.1])

Input: A 128-bit cipher key k, and a 128-bit plaintext message m
Output: A 128-bit ciphertext message $c = \text{AES-128.Enc}(k, m)$

$s \leftarrow m$
$s \leftarrow \texttt{AddRoundKey}(s, k = rk^{(0)})$
for $r = 1$ **upto** $Nr - 1$ **step** $+1$ **do**
  $k \leftarrow \texttt{EvolveRoundKey}(k, rc^{(r)})$
  $s \leftarrow \texttt{SubBytes}(s)$
  $s \leftarrow \texttt{ShiftRows}(s)$
  $s \leftarrow \texttt{MixColumns}(s)$
  $s \leftarrow \texttt{AddRoundKey}(s, k = rk^{(r)})$
end
$k \leftarrow \texttt{EvolveRoundKey}(k, rc^{(10)})$
$s \leftarrow \texttt{SubBytes}(s)$
$s \leftarrow \texttt{ShiftRows}(s)$
$s \leftarrow \texttt{AddRoundKey}(s, k = rk^{(10)})$
$c \leftarrow s$
return c

Part 2: in practice (1)

- Challenge:
  - given a functionality "stack", i.e.,



  bridge gap between what we have (bottom) and want (top),

  - an **implementation strategy** for doing so must consider many
    - goals : parameter set, functionality, ...
    - metrics : latency, throughput, memory footprint, ...
    - constraints : hardware versus software, data-path width, ...

    :

- Strategy #1 [2, Section 4.1]:
  - use pure software (i.e., via ISA),
  - adopt an unpacked representation of state and round key matrices, using (an array of 16) 8-bit bytes, i.e., instances of type `uint8_t`,
  - favour use of iterative (i.e., rolled) loops,
  - compute (or evolve) round keys,
  - compute round functions.

- Selective pre-computation can be effective

$$\begin{aligned}
\text{multiplication-by-}\mathbf{x} &: \mathbb{F}_{2^8} \to \mathbb{F}_{2^8} \rightsquigarrow \text{256B look-up table} \\
\text{division-by-}\mathbf{x} &: \mathbb{F}_{2^8} \to \mathbb{F}_{2^8} \rightsquigarrow \text{256B look-up table} \\
\text{round constants} &: \mathbb{Z} \to \mathbb{F}_{2^8} \rightsquigarrow \mathit{Nr}\text{B look-up table} \\
\text{S-box} &: \mathbb{F}_{2^8} \to \mathbb{F}_{2^8} \rightsquigarrow \text{256B look-up table}
\end{aligned}$$

noting that various (sub-)options exist for the S-box, e.g.,

Pre-compute	Compute	Footprint	Applicability	
			Enc	Dec
	$f \circ g$	0B	✓	✓
g	f	256B	✓	✓
f	g	256B	✓	×
$f \circ g$		256B	✓	×

Listing

```
1 void aes_enc_rnd_key( uint8_t* s, const uint8_t* rk ) {
2   for( int i = 0; i < 16; i++ ) {
3     s[ i ] = s[ i ] ^ rk[ i ];
4   }
5 }
```

Listing

```
1 void aes_enc_rnd_sub( uint8_t* s ) {
2   for( int i = 0; i < 16; i++ ) {
3     s[ i ] = aes_enc_sbox( s[ i ] );
4   }
5 }
```

Listing

```
1 void aes_enc_rnd_row( uint8_t* s ) {
2   AES_ENC_RND_ROW_STEP( 0x1, 0x5, 0x9, 0xD,
3                         0xD, 0x1, 0x5, 0x9 );
4   AES_ENC_RND_ROW_STEP( 0x2, 0x6, 0xA, 0xE,
5                         0xA, 0xE, 0x2, 0x6 );
6   AES_ENC_RND_ROW_STEP( 0x3, 0x7, 0xB, 0xF,
7                         0x7, 0xB, 0xF, 0x3 );
8 }
```

Listing

```
1  #define AES_ENC_RND_ROW_STEP(a,b,c,d,e,f,g,h) {        \
2    uint8_t __a1 = s[ a ];                               \
3    uint8_t __b1 = s[ b ];                               \
4    uint8_t __c1 = s[ c ];                               \
5    uint8_t __d1 = s[ d ];                               \
6                                                         \
7    s[ e ] = __a1;                                       \
8    s[ f ] = __b1;                                       \
9    s[ g ] = __c1;                                       \
10   s[ h ] = __d1;                                       \
11 }
```

Listing

```
1 void aes_enc_rnd_mix( uint8_t* s ) {
2   for( int i = 0; i < 4; i++, s += 4 ) {
3     AES_ENC_RND_MIX_STEP( 0x0, 0x1, 0x2, 0x3 );
4   }
5 }
```

Listing

```
1  #define AES_ENC_RND_MIX_STEP(a,b,c,d) {                  \
2    uint8_t __a1 = s[ a ], __a2 = aes_gf28_mulx( __a1 ); \
3    uint8_t __b1 = s[ b ], __b2 = aes_gf28_mulx( __b1 ); \
4    uint8_t __c1 = s[ c ], __c2 = aes_gf28_mulx( __c1 ); \
5    uint8_t __d1 = s[ d ], __d2 = aes_gf28_mulx( __d1 ); \
6                                                         \
7    uint8_t __a3 = __a1 ^ __a2;                          \
8    uint8_t __b3 = __b1 ^ __b2;                          \
9    uint8_t __c3 = __c1 ^ __c2;                          \
10   uint8_t __d3 = __d1 ^ __d2;                          \
11                                                        \
12   s[ a ] = __a2 ^ __b3 ^ __c1 ^ __d1;                  \
13   s[ b ] = __a1 ^ __b2 ^ __c3 ^ __d1;                  \
14   s[ c ] = __a1 ^ __b1 ^ __c2 ^ __d3;                  \
15   s[ d ] = __a3 ^ __b1 ^ __c1 ^ __d2;                  \
16 }
```

Listing

```
1  void aes_enc( uint8_t* c, const uint8_t* m, const uint8_t* k ) {
2      uint8_t  s[ 4 * AES_128_NB ];
3      uint8_t rk[ 4 * AES_128_NB ];
4
5      memcpy(  s, m, ( 4 * AES_128_NB ) * sizeof( uint8_t ) );
6      memcpy( rk, k, ( 4 * AES_128_NB ) * sizeof( uint8_t ) );
7
8    //      1 initial   round
9      aes_enc_rnd_key( s, rk );
10   // Nr - 1 interated rounds
11   for( int r = 1; r < AES_128_NR; r++ ) {
12     aes_enc_key_evolve( rk, rk, aes_rcon(          r ) );
13     aes_enc_rnd_sub( s     );
14     aes_enc_rnd_row( s     );
15     aes_enc_rnd_mix( s     );
16     aes_enc_rnd_key( s, rk );
17   }
18   //      1 final     round
19     aes_enc_key_evolve( rk, rk, aes_rcon( AES_128_NR ) );
20     aes_enc_rnd_sub( s     );
21     aes_enc_rnd_row( s     );
22     aes_enc_rnd_key( s, rk );
23
24     memcpy(  c, s, ( 4 * AES_128_NB ) * sizeof( uint8_t ) );
25 }
```

- Strategy #2 [2, Section 4.2] (aka. T-tables):
  - use pure software (i.e., via ISA),
  - adopt a column-packed representation of state and round key matrices, using (a set of 4) 32-bit words, i.e., instances of type `uint32_t`,
  - favour use of straight-line (i.e., unrolled) loops,
  - pre-compute (or expand) round keys,
  - pre-compute round functions.

- Idea: pre-compute most of the round, i.e.,

$$\texttt{MixColumns} \circ \texttt{ShiftRows} \circ \texttt{SubBytes}.$$

- Idea: pre-compute most of the round, i.e.,

$$\texttt{MixColumns} \circ \texttt{ShiftRows} \circ \texttt{SubBytes}.$$

- Step #1: we already know applying `MixColumns` will yield

$$\begin{bmatrix} s'^{(r)}_{0,j} \\ s'^{(r)}_{1,j} \\ s'^{(r)}_{2,j} \\ s'^{(r)}_{3,j} \end{bmatrix} = \begin{bmatrix} \texttt{02} & \texttt{03} & \texttt{01} & \texttt{01} \\ \texttt{01} & \texttt{02} & \texttt{03} & \texttt{01} \\ \texttt{01} & \texttt{01} & \texttt{02} & \texttt{03} \\ \texttt{03} & \texttt{01} & \texttt{01} & \texttt{02} \end{bmatrix} \otimes_{\mathbb{F}_{2^8}} \begin{bmatrix} s^{(r)}_{0,j} \\ s^{(r)}_{1,j} \\ s^{(r)}_{2,j} \\ s^{(r)}_{3,j} \end{bmatrix}$$

- Idea: pre-compute most of the round, i.e.,

$$\texttt{MixColumns} \circ \texttt{ShiftRows} \circ \texttt{SubBytes}.$$

- Step #1: we already know applying `MixColumns` will yield

$$\begin{bmatrix} s'^{(r)}_{0,j} \\ s'^{(r)}_{1,j} \\ s'^{(r)}_{2,j} \\ s'^{(r)}_{3,j} \end{bmatrix} = \begin{bmatrix} (\texttt{02} \otimes_{\mathbb{F}_{2^8}} s^{(r)}_{0,j}) \oplus_{\mathbb{F}_{2^8}} (\texttt{03} \otimes_{\mathbb{F}_{2^8}} s^{(r)}_{1,j}) \oplus_{\mathbb{F}_{2^8}} (\texttt{01} \otimes_{\mathbb{F}_{2^8}} s^{(r)}_{2,j}) \oplus_{\mathbb{F}_{2^8}} (\texttt{01} \otimes_{\mathbb{F}_{2^8}} s^{(r)}_{3,j}) \\ (\texttt{01} \otimes_{\mathbb{F}_{2^8}} s^{(r)}_{0,j}) \oplus_{\mathbb{F}_{2^8}} (\texttt{02} \otimes_{\mathbb{F}_{2^8}} s^{(r)}_{1,j}) \oplus_{\mathbb{F}_{2^8}} (\texttt{03} \otimes_{\mathbb{F}_{2^8}} s^{(r)}_{2,j}) \oplus_{\mathbb{F}_{2^8}} (\texttt{01} \otimes_{\mathbb{F}_{2^8}} s^{(r)}_{3,j}) \\ (\texttt{01} \otimes_{\mathbb{F}_{2^8}} s^{(r)}_{0,j}) \oplus_{\mathbb{F}_{2^8}} (\texttt{01} \otimes_{\mathbb{F}_{2^8}} s^{(r)}_{1,j}) \oplus_{\mathbb{F}_{2^8}} (\texttt{02} \otimes_{\mathbb{F}_{2^8}} s^{(r)}_{2,j}) \oplus_{\mathbb{F}_{2^8}} (\texttt{03} \otimes_{\mathbb{F}_{2^8}} s^{(r)}_{3,j}) \\ (\texttt{03} \otimes_{\mathbb{F}_{2^8}} s^{(r)}_{0,j}) \oplus_{\mathbb{F}_{2^8}} (\texttt{01} \otimes_{\mathbb{F}_{2^8}} s^{(r)}_{1,j}) \oplus_{\mathbb{F}_{2^8}} (\texttt{01} \otimes_{\mathbb{F}_{2^8}} s^{(r)}_{2,j}) \oplus_{\mathbb{F}_{2^8}} (\texttt{02} \otimes_{\mathbb{F}_{2^8}} s^{(r)}_{3,j}) \end{bmatrix}$$

- Idea: pre-compute most of the round, i.e.,

$$\texttt{MixColumns} \circ \texttt{ShiftRows} \circ \texttt{SubBytes}.$$

- Step #1: we already know applying `MixColumns` will yield

$$\begin{bmatrix} s'^{(r)}_{0,j} \\ s'^{(r)}_{1,j} \\ s'^{(r)}_{2,j} \\ s'^{(r)}_{3,j} \end{bmatrix} = \begin{bmatrix} \texttt{02} \otimes_{\mathbb{F}_{2^8}} s^{(r)}_{0,j} \\ \texttt{01} \otimes_{\mathbb{F}_{2^8}} s^{(r)}_{0,j} \\ \texttt{01} \otimes_{\mathbb{F}_{2^8}} s^{(r)}_{0,j} \\ \texttt{03} \otimes_{\mathbb{F}_{2^8}} s^{(r)}_{0,j} \end{bmatrix} \oplus_{\mathbb{F}_{2^8}} \begin{bmatrix} \texttt{03} \otimes_{\mathbb{F}_{2^8}} s^{(r)}_{1,j} \\ \texttt{02} \otimes_{\mathbb{F}_{2^8}} s^{(r)}_{1,j} \\ \texttt{01} \otimes_{\mathbb{F}_{2^8}} s^{(r)}_{1,j} \\ \texttt{01} \otimes_{\mathbb{F}_{2^8}} s^{(r)}_{1,j} \end{bmatrix} \oplus_{\mathbb{F}_{2^8}} \begin{bmatrix} \texttt{01} \otimes_{\mathbb{F}_{2^8}} s^{(r)}_{2,j} \\ \texttt{03} \otimes_{\mathbb{F}_{2^8}} s^{(r)}_{2,j} \\ \texttt{02} \otimes_{\mathbb{F}_{2^8}} s^{(r)}_{2,j} \\ \texttt{01} \otimes_{\mathbb{F}_{2^8}} s^{(r)}_{2,j} \end{bmatrix} \oplus_{\mathbb{F}_{2^8}} \begin{bmatrix} \texttt{01} \otimes_{\mathbb{F}_{2^8}} s^{(r)}_{3,j} \\ \texttt{01} \otimes_{\mathbb{F}_{2^8}} s^{(r)}_{3,j} \\ \texttt{03} \otimes_{\mathbb{F}_{2^8}} s^{(r)}_{3,j} \\ \texttt{02} \otimes_{\mathbb{F}_{2^8}} s^{(r)}_{3,j} \end{bmatrix}$$

- Idea: pre-compute most of the round, i.e.,

$$\texttt{MixColumns} \circ \texttt{ShiftRows} \circ \texttt{SubBytes}.$$

- Step #2: an equivalent RHS can be formed from pre-computed look-up tables, i.e.,

$$\begin{bmatrix} s'^{(r)}_{0,j} \\ s'^{(r)}_{1,j} \\ s'^{(r)}_{2,j} \\ s'^{(r)}_{3,j} \end{bmatrix} = \begin{matrix} T_0[s^{(r)}_{0,j}] & \oplus_{\mathbb{F}_{2^8}} & T_1[s^{(r)}_{1,j}] & \oplus_{\mathbb{F}_{2^8}} \\ T_2[s^{(r)}_{2,j}] & \oplus_{\mathbb{F}_{2^8}} & T_3[s^{(r)}_{3,j}] & \end{matrix}$$

where

$$T_0[x] = \begin{bmatrix} \texttt{02} \otimes_{\mathbb{F}_{2^8}} x \\ \texttt{01} \otimes_{\mathbb{F}_{2^8}} x \\ \texttt{01} \otimes_{\mathbb{F}_{2^8}} x \\ \texttt{03} \otimes_{\mathbb{F}_{2^8}} x \end{bmatrix} \quad T_1[x] = \begin{bmatrix} \texttt{03} \otimes_{\mathbb{F}_{2^8}} x \\ \texttt{02} \otimes_{\mathbb{F}_{2^8}} x \\ \texttt{01} \otimes_{\mathbb{F}_{2^8}} x \\ \texttt{01} \otimes_{\mathbb{F}_{2^8}} x \end{bmatrix}$$

$$T_2[x] = \begin{bmatrix} \texttt{01} \otimes_{\mathbb{F}_{2^8}} x \\ \texttt{03} \otimes_{\mathbb{F}_{2^8}} x \\ \texttt{02} \otimes_{\mathbb{F}_{2^8}} x \\ \texttt{01} \otimes_{\mathbb{F}_{2^8}} x \end{bmatrix} \quad T_3[x] = \begin{bmatrix} \texttt{01} \otimes_{\mathbb{F}_{2^8}} x \\ \texttt{01} \otimes_{\mathbb{F}_{2^8}} x \\ \texttt{03} \otimes_{\mathbb{F}_{2^8}} x \\ \texttt{02} \otimes_{\mathbb{F}_{2^8}} x \end{bmatrix}$$

- Idea: pre-compute most of the round, i.e.,

$$\texttt{MixColumns} \circ \texttt{ShiftRows} \circ \texttt{SubBytes}.$$

- Step #3: `ShiftRows` and `SubBytes` can then be folded into the tables and look-ups, i.e.,

$$\begin{bmatrix} s'^{(r)}_{0,j} \\ s'^{(r)}_{1,j} \\ s'^{(r)}_{2,j} \\ s'^{(r)}_{3,j} \end{bmatrix} = \begin{matrix} T_0[s^{(r)}_{0,j+0 \pmod{Nr}}] & \oplus_{\mathbb{F}_{2^8}} & T_1[s^{(r)}_{1,j+1 \pmod{Nr}}] & \oplus_{\mathbb{F}_{2^8}} \\ T_2[s^{(r)}_{2,j+2 \pmod{Nr}}] & \oplus_{\mathbb{F}_{2^8}} & T_3[s^{(r)}_{3,j+3 \pmod{Nr}}] \end{matrix}$$

where

$$T_0[x] = \begin{bmatrix} \texttt{02} \otimes_{\mathbb{F}_{2^8}} \text{S-BOX}(x) \\ \texttt{01} \otimes_{\mathbb{F}_{2^8}} \text{S-BOX}(x) \\ \texttt{01} \otimes_{\mathbb{F}_{2^8}} \text{S-BOX}(x) \\ \texttt{03} \otimes_{\mathbb{F}_{2^8}} \text{S-BOX}(x) \end{bmatrix} \quad T_1[x] = \begin{bmatrix} \texttt{03} \otimes_{\mathbb{F}_{2^8}} \text{S-BOX}(x) \\ \texttt{02} \otimes_{\mathbb{F}_{2^8}} \text{S-BOX}(x) \\ \texttt{01} \otimes_{\mathbb{F}_{2^8}} \text{S-BOX}(x) \\ \texttt{01} \otimes_{\mathbb{F}_{2^8}} \text{S-BOX}(x) \end{bmatrix}$$

$$T_2[x] = \begin{bmatrix} \texttt{01} \otimes_{\mathbb{F}_{2^8}} \text{S-BOX}(x) \\ \texttt{03} \otimes_{\mathbb{F}_{2^8}} \text{S-BOX}(x) \\ \texttt{02} \otimes_{\mathbb{F}_{2^8}} \text{S-BOX}(x) \\ \texttt{01} \otimes_{\mathbb{F}_{2^8}} \text{S-BOX}(x) \end{bmatrix} \quad T_3[x] = \begin{bmatrix} \texttt{01} \otimes_{\mathbb{F}_{2^8}} \text{S-BOX}(x) \\ \texttt{01} \otimes_{\mathbb{F}_{2^8}} \text{S-BOX}(x) \\ \texttt{03} \otimes_{\mathbb{F}_{2^8}} \text{S-BOX}(x) \\ \texttt{02} \otimes_{\mathbb{F}_{2^8}} \text{S-BOX}(x) \end{bmatrix}$$

Listing

```
1 #define AES_ENC_RND_INIT() {                                                \
2   t_0 = rkp[ 0 ] ^ s_0;                                                     \
3   t_1 = rkp[ 1 ] ^ s_1;                                                     \
4   t_2 = rkp[ 2 ] ^ s_2;                                                     \
5   t_3 = rkp[ 3 ] ^ s_3;                                                     \
6                                                                             \
7   rkp += AES_128_NB; s_0 = t_0; s_1 = t_1; s_2 = t_2; s_3 = t_3;           \
8 }
```

Listing

```
#define AES_ENC_RND_ITER() {                                                    \
  t_0 = rkp[ 0 ] ^ ( AES_ENC_TBOX_0[ ( s_0 >>  0 ) & 0xFF ]                ) ^ \
                  ( AES_ENC_TBOX_1[ ( s_1 >>  8 ) & 0xFF ]                ) ^ \
                  ( AES_ENC_TBOX_2[ ( s_2 >> 16 ) & 0xFF ]                ) ^ \
                  ( AES_ENC_TBOX_3[ ( s_3 >> 24 ) & 0xFF ]                ) ; \
  t_1 = rkp[ 1 ] ^ ( AES_ENC_TBOX_0[ ( s_1 >>  0 ) & 0xFF ]                ) ^ \
                  ( AES_ENC_TBOX_1[ ( s_2 >>  8 ) & 0xFF ]                ) ^ \
                  ( AES_ENC_TBOX_2[ ( s_3 >> 16 ) & 0xFF ]                ) ^ \
                  ( AES_ENC_TBOX_3[ ( s_0 >> 24 ) & 0xFF ]                ) ; \
  t_2 = rkp[ 2 ] ^ ( AES_ENC_TBOX_0[ ( s_2 >>  0 ) & 0xFF ]                ) ^ \
                  ( AES_ENC_TBOX_1[ ( s_3 >>  8 ) & 0xFF ]                ) ^ \
                  ( AES_ENC_TBOX_2[ ( s_0 >> 16 ) & 0xFF ]                ) ^ \
                  ( AES_ENC_TBOX_3[ ( s_1 >> 24 ) & 0xFF ]                ) ; \
  t_3 = rkp[ 3 ] ^ ( AES_ENC_TBOX_0[ ( s_3 >>  0 ) & 0xFF ]                ) ^ \
                  ( AES_ENC_TBOX_1[ ( s_0 >>  8 ) & 0xFF ]                ) ^ \
                  ( AES_ENC_TBOX_2[ ( s_1 >> 16 ) & 0xFF ]                ) ^ \
                  ( AES_ENC_TBOX_3[ ( s_2 >> 24 ) & 0xFF ]                ) ; \
                                                                              \
  rkp += AES_128_NB; s_0 = t_0; s_1 = t_1; s_2 = t_2; s_3 = t_3;              \
}
```

Listing

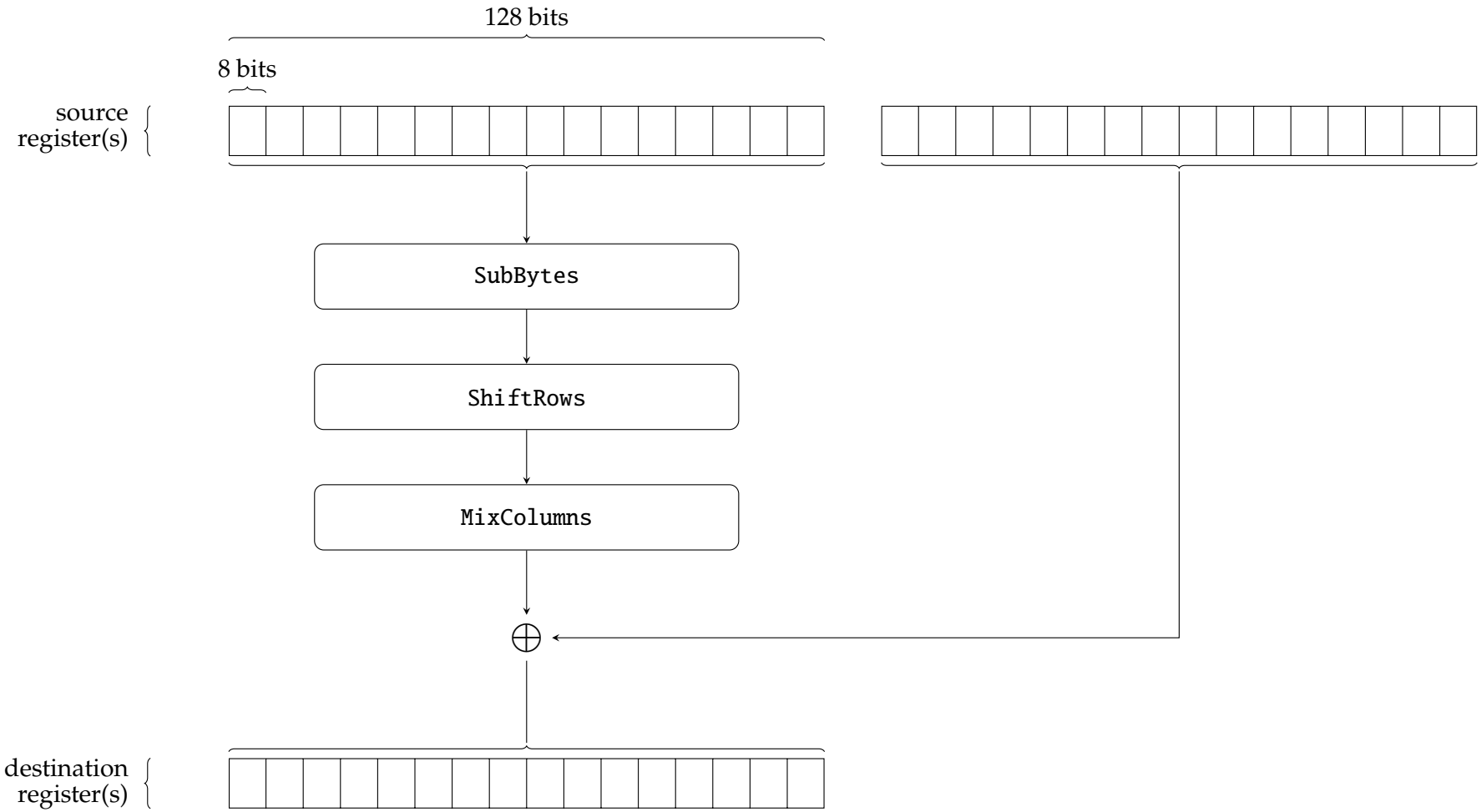
```
#define AES_ENC_RND_FINI() {                                                    \
  t_0 = rkp[ 0 ] ^ ( AES_ENC_TBOX_4[ ( s_0 >>  0 ) & 0xFF ] & 0x000000FF ) ^ \
                   ( AES_ENC_TBOX_4[ ( s_1 >>  8 ) & 0xFF ] & 0x0000FF00 ) ^ \
                   ( AES_ENC_TBOX_4[ ( s_2 >> 16 ) & 0xFF ] & 0x00FF0000 ) ^ \
                   ( AES_ENC_TBOX_4[ ( s_3 >> 24 ) & 0xFF ] & 0xFF000000 ) ; \
  t_1 = rkp[ 1 ] ^ ( AES_ENC_TBOX_4[ ( s_1 >>  0 ) & 0xFF ] & 0x000000FF ) ^ \
                   ( AES_ENC_TBOX_4[ ( s_2 >>  8 ) & 0xFF ] & 0x0000FF00 ) ^ \
                   ( AES_ENC_TBOX_4[ ( s_3 >> 16 ) & 0xFF ] & 0x00FF0000 ) ^ \
                   ( AES_ENC_TBOX_4[ ( s_0 >> 24 ) & 0xFF ] & 0xFF000000 ) ; \
  t_2 = rkp[ 2 ] ^ ( AES_ENC_TBOX_4[ ( s_2 >>  0 ) & 0xFF ] & 0x000000FF ) ^ \
                   ( AES_ENC_TBOX_4[ ( s_3 >>  8 ) & 0xFF ] & 0x0000FF00 ) ^ \
                   ( AES_ENC_TBOX_4[ ( s_0 >> 16 ) & 0xFF ] & 0x00FF0000 ) ^ \
                   ( AES_ENC_TBOX_4[ ( s_1 >> 24 ) & 0xFF ] & 0xFF000000 ) ; \
  t_3 = rkp[ 3 ] ^ ( AES_ENC_TBOX_4[ ( s_3 >>  0 ) & 0xFF ] & 0x000000FF ) ^ \
                   ( AES_ENC_TBOX_4[ ( s_0 >>  8 ) & 0xFF ] & 0x0000FF00 ) ^ \
                   ( AES_ENC_TBOX_4[ ( s_1 >> 16 ) & 0xFF ] & 0x00FF0000 ) ^ \
                   ( AES_ENC_TBOX_4[ ( s_2 >> 24 ) & 0xFF ] & 0xFF000000 ) ; \
                                                                              \
  rkp += AES_128_NB; s_0 = t_0; s_1 = t_1; s_2 = t_2; s_3 = t_3;              \
}
```

Listing

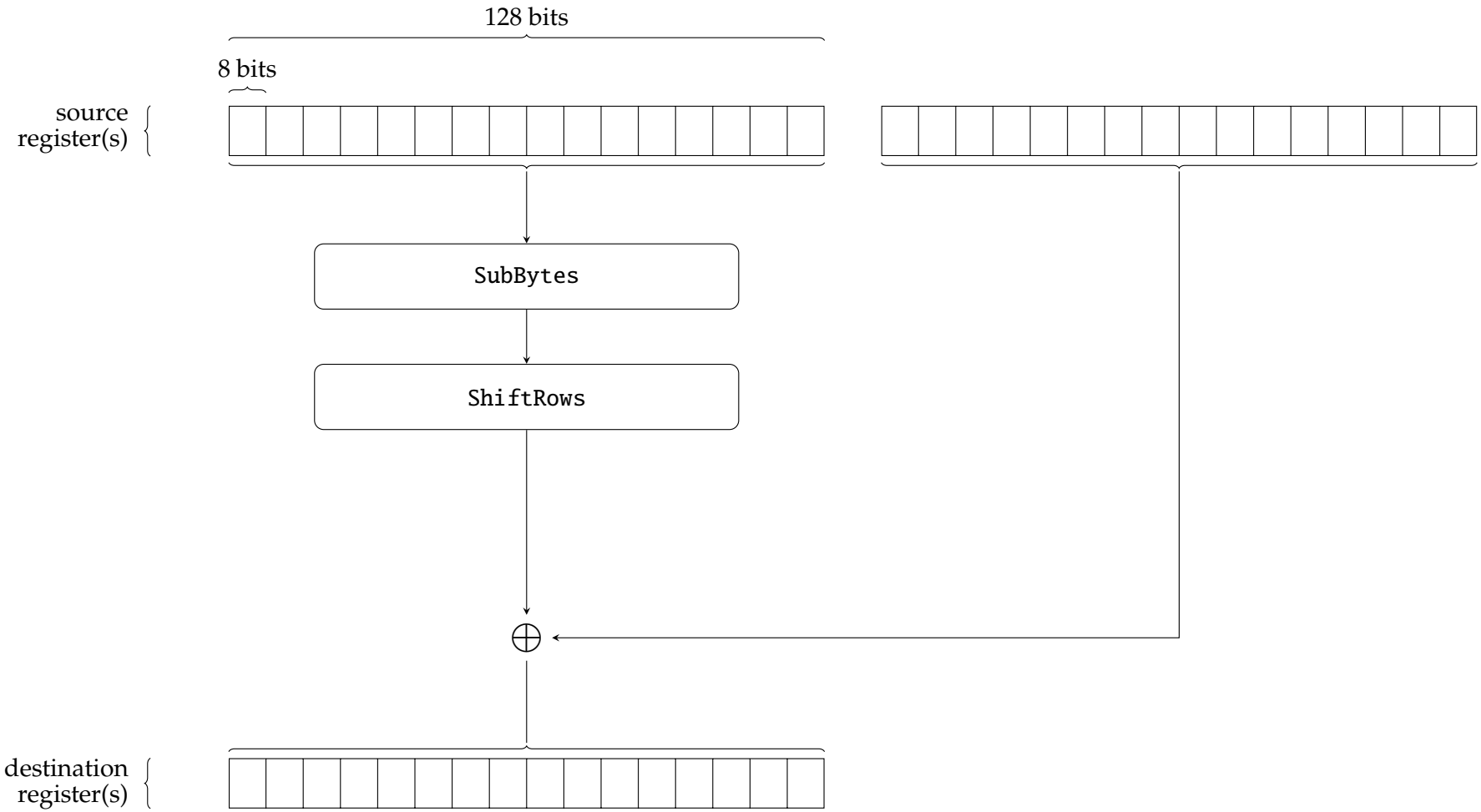
```
1  void aes_enc( uint8_t* c, const uint8_t* m, const uint8_t* rk ) {
2    uint32_t s_0, s_1, s_2, s_3, t_0, t_1, t_2, t_3;
3
4    U8_TO_U32_LE( s_0, m,  0 ); U8_TO_U32_LE( s_1, m,  4 );
5    U8_TO_U32_LE( s_2, m,  8 ); U8_TO_U32_LE( s_3, m, 12 );
6
7    uint32_t *rkp = ( uint32_t* )( rk );
8
9    //      1 initial   round
10     AES_ENC_RND_INIT();
11   // Nr - 1 interated rounds
12   for( int i = 1; i < AES_128_NR; i++ ) {
13     AES_ENC_RND_ITER();
14   }
15   //      1 final     round
16     AES_ENC_RND_FINI();
17
18   U32_TO_U8_LE( c, s_0,  0 ); U32_TO_U8_LE( c, s_1,  4 );
19   U32_TO_U8_LE( c, s_2,  8 ); U32_TO_U8_LE( c, s_3, 12 );
20 }
```

- Strategy #3 [7] (i.e., Intel AES-NI):
  - use hybrid of software (i.e., via ISA) and hardware (i.e., via ISE),
  - adopt a fully-packed representation of state and round key matrices, using 128-bit words, i.e., instances of type `__m128i`,
  - favour use of straight-line (i.e., unrolled) loops,
  - pre-compute (or expand) round keys,
  - compute round functions.

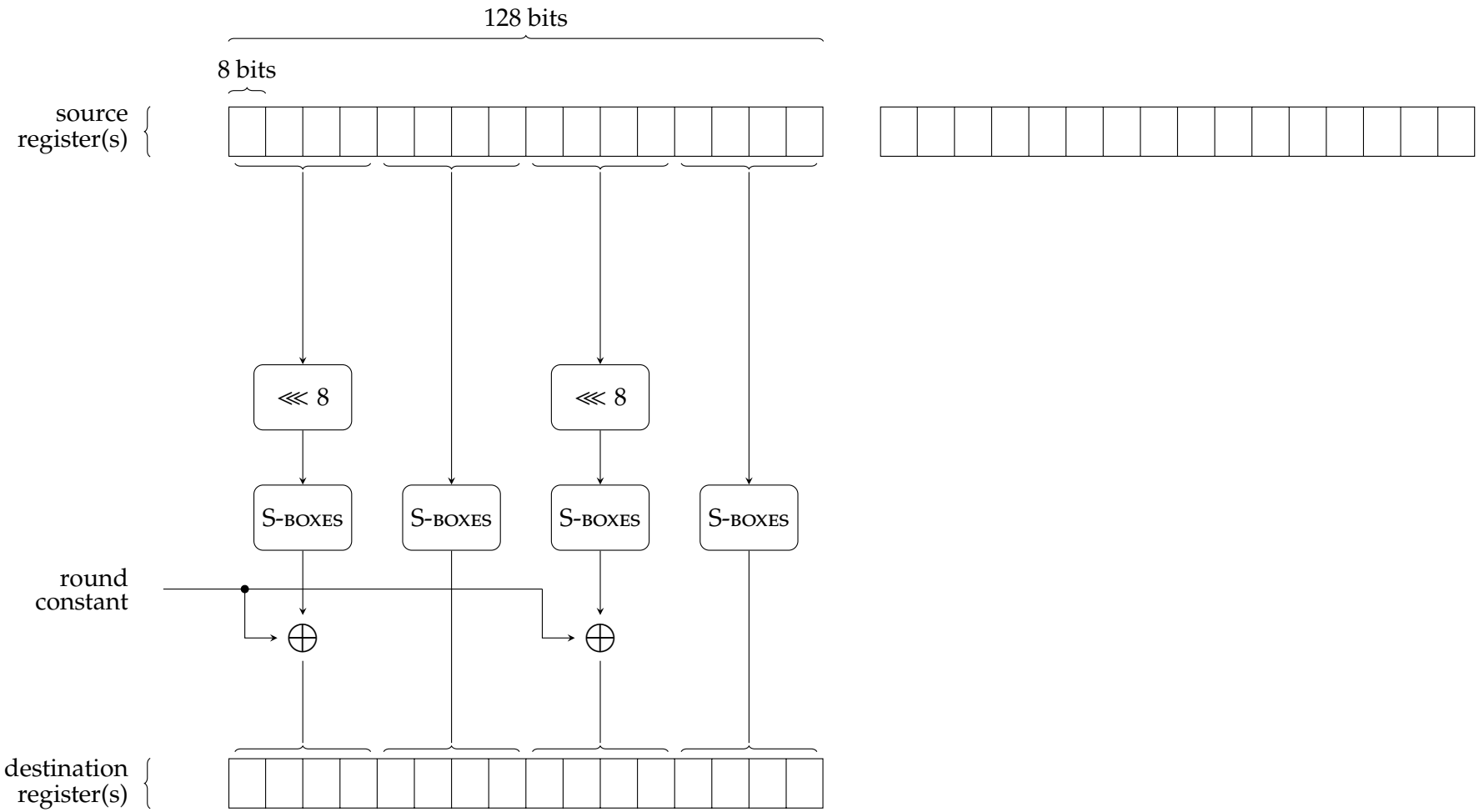
Circuit (aesenc [6, Pages 3-54–3-55])



Circuit (`aesenclast` [6, Pages 3-56–3-57])

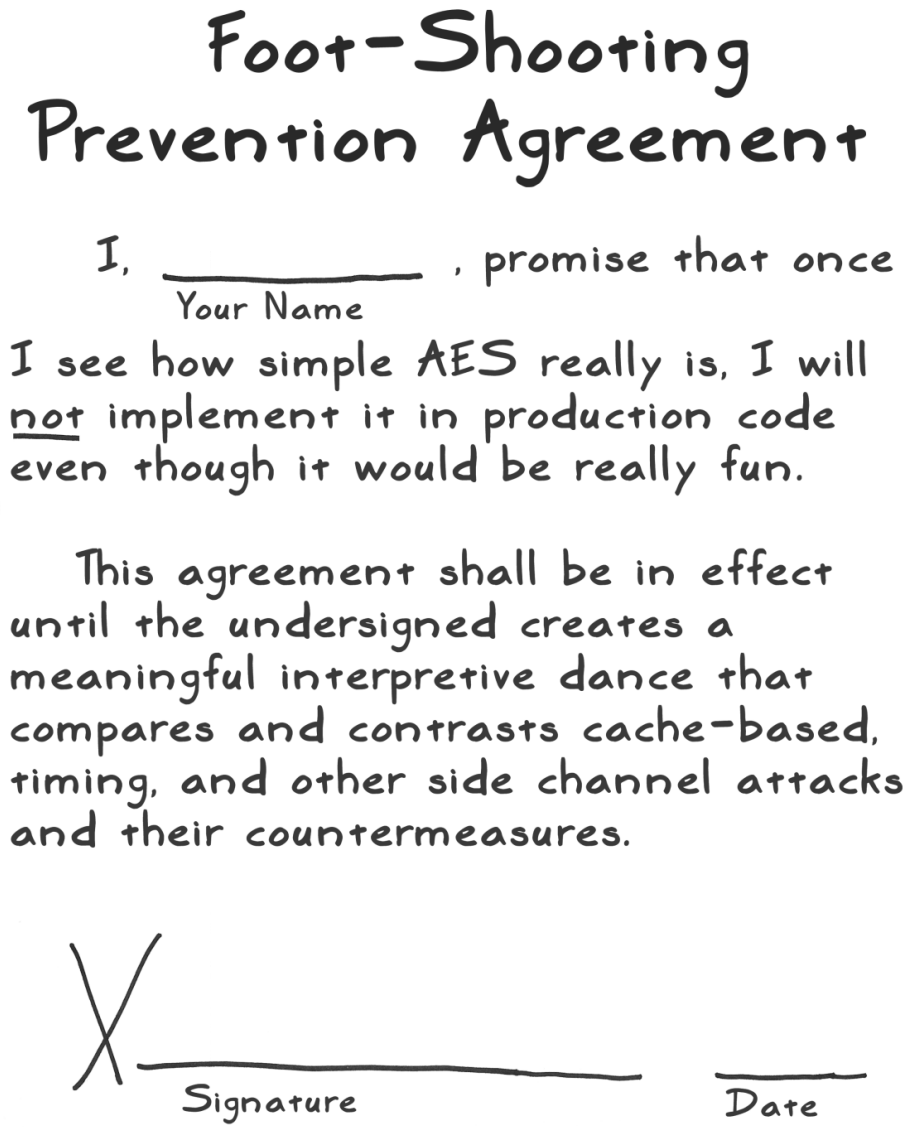


Circuit (`aeskeygenassist` [6, Pages 3-59–3-60])



Listing

```
1  void aes_enc( uint8_t* c, const uint8_t* m, const uint8_t* rk ) {
2    __m128i s = _mm_load_si128( ( __m128i* )( m ) );
3    __m128i* rkp = ( __m128i* )( rk );
4
5    //      1 initial  round
6    s =        _mm_xor_si128( s, _mm_load_si128( rkp++ ) );
7    // Nr - 1 iterated rounds
8    s =     _mm_aesenc_si128( s, _mm_load_si128( rkp++ ) );
9    s =     _mm_aesenc_si128( s, _mm_load_si128( rkp++ ) );
10   s =     _mm_aesenc_si128( s, _mm_load_si128( rkp++ ) );
11   s =     _mm_aesenc_si128( s, _mm_load_si128( rkp++ ) );
12   s =     _mm_aesenc_si128( s, _mm_load_si128( rkp++ ) );
13   s =     _mm_aesenc_si128( s, _mm_load_si128( rkp++ ) );
14   s =     _mm_aesenc_si128( s, _mm_load_si128( rkp++ ) );
15   s =     _mm_aesenc_si128( s, _mm_load_si128( rkp++ ) );
16   s =     _mm_aesenc_si128( s, _mm_load_si128( rkp++ ) );
17   //      1 final    round
18   s = _mm_aesenclast_si128( s, _mm_load_si128( rkp++ ) );
19
20   _mm_store_si128( ( __m128i* )( c ), s );
21 }
```



https://www.moserware.com/assets/stick-figure-guide-to-advanced/aes_act_3_scene_02_agreement_1100.png

Conclusions

- Take away points: you can often simple *use*

$$c = \text{AES-128.Enc}(k, m),$$

but understanding internals of this primitive can be useful and/or important.

- some historically interesting aspects; some "portable" concepts,
- close relationship between primitive and underlying Mathematics,
- wide range of viable implementation strategies,
- extensive deployment, in various contexts and use-cases.

Additional Reading

- *Wikipedia: Advanced Encryption Standard (AES).* URL: https://en.wikipedia.org/wiki/Advanced_Encryption_Standard.
- *Advanced Encryption Standard (AES).* National Institute of Standards and Technology (NIST) Federal Information Processing Standard (FIPS) 197-upd1. 2023. URL: https://doi.org/10.6028/NIST.FIPS.197-upd1.
- L.R. Knudsen and M.J.B. Robshaw. *The Block Cipher Companion.* Springer, 2011.
- J. Daemen and V. Rijmen. *The Design of Rijndael.* Springer, 2002.

References

[1] *Wikipedia: Advanced Encryption Standard (AES)*. URL: https://en.wikipedia.org/wiki/Advanced_Encryption_Standard (see p. 45).

[2] J. Daemen and V. Rijmen. *The Design of Rijndael*. Springer, 2002 (see pp. 1, 20, 27, 45).

[3] L.R. Knudsen and M.J.B. Robshaw. *The Block Cipher Companion*. Springer, 2011 (see p. 45).

[4] J. Daemen, L. Knudsen, and V. Rijmen. "The block cipher Square". In: *Fast Software Encryption (FSE)*. LNCS 1267. Springer-Verlag, 1997, pp. 149–165 (see p. 1).

[5] J. Daemen and V. Rijmen. "The Block Cipher Rijndael". In: *Smart Card Research and Applications (CARDIS)*. LNCS 1820. Springer-Verlag, 1998, pp. 277–284 (see p. 1).

[6] *Intel 64 and IA-32 architectures – Software Developer's Manual (Volume 2: Instruction Set Reference A-Z)*. Tech. rep. 325383-071US. Intel Corp., 2019. URL: http://software.intel.com/en-us/articles/intel-sdm (see pp. 39–41).

[7] *Intel Advanced Encryption Standard (AES) Instructions Set*. Tech. rep. Intel Corp., 2012. URL: http://software.intel.com/sites/default/files/article/165683/aes-wp-2012-09-22-v01.pdf (see p. 38).

[8] *Advanced Encryption Standard (AES)*. National Institute of Standards and Technology (NIST) Federal Information Processing Standard (FIPS) 197-upd1. 2023. URL: https://doi.org/10.6028/NIST.FIPS.197-upd1 (see pp. 1, 3, 4, 8–18, 45).