Applied Cryptology

Daniel Page

Department of Computer Science,
University Of Bristol,
Merchant Venturers Building,
Woodland Road,
Bristol, BS8 1UB. UK.
⟨csdsp@bristol.ac.uk⟩

April 24, 2024

Keep in mind there are *two* PDFs available (of which this is the latter):

1. a PDF of examinable material used as lecture slides, and
2. a PDF of non-examinable, extra material:
   - the associated notes page may be pre-populated with extra, written explaination of material covered in lecture(s), plus
   - anything with a “grey’ed out” header/footer represents extra material which is useful and/or interesting but out of scope (and hence not covered).

Notes:

Notes:

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

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

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

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

Notes:

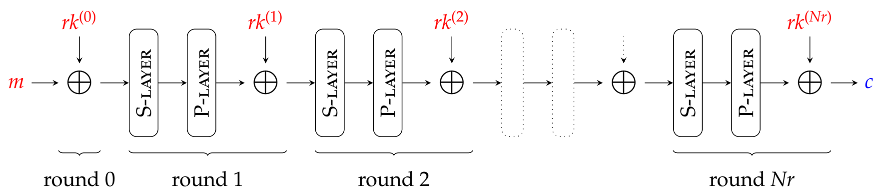
Part 1: in theory (1)

Specification

- AES is an (iterated) block cipher, where

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

  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.

Notes:

Part 1: in theory (2)
Specification

- 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.

Notes:

Part 1: in theory (3)
Low(er)-level concepts/components

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

Notes:

Part 1: in theory (4)

Low(er)-level concepts/components

- 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.,

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

Notes:

Part 1: in theory (4)

Low(er)-level concepts/components

- 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.,

$$\begin{array}{lclclclcl} r & = & \texttt{01} \otimes_{\mathbb{F}_{2^8}} x & = & 1 & \otimes_{\mathbb{F}_{2^8}} x & = & & x \\ r & = & \texttt{02} \otimes_{\mathbb{F}_{2^8}} x & = & \mathbf{x} & \otimes_{\mathbb{F}_{2^8}} x & = & \texttt{xtimes}(x) & \\ r & = & \texttt{03} \otimes_{\mathbb{F}_{2^8}} x & = & (\mathbf{x} + 1) & \otimes_{\mathbb{F}_{2^8}} x & = & \texttt{xtimes}(x) \oplus & x \end{array}$$

Notes:

Part 1: in theory (4)

Low(er)-level concepts/components

- 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.,
       $$r \;=\; \texttt{xtimes}(x) \;=\; \mathbf{x} \otimes_{\mathbb{F}_{2^8}} x \;\mapsto\; \begin{cases} (\, x \ll 1\,) \oplus \texttt{11B} & \text{if } x_7 = 1 \\ x \ll 1 & \text{otherwise} \end{cases}$$
    3. **multiplication-by-c**, e.g.,
       $$\begin{array}{lclclcl} r & = & \texttt{01} \otimes_{\mathbb{F}_{2^8}} x & = & 1 \otimes_{\mathbb{F}_{2^8}} x & = & x \\ r & = & \texttt{02} \otimes_{\mathbb{F}_{2^8}} x & = & \mathbf{x} \otimes_{\mathbb{F}_{2^8}} x & = & \texttt{xtimes}(x) \\ r & = & \texttt{03} \otimes_{\mathbb{F}_{2^8}} x & = & (\,\mathbf{x}+1\,) \otimes_{\mathbb{F}_{2^8}} x & = & \texttt{xtimes}(x) \oplus x \end{array}$$

Notes:

Part 1: in theory (5)

Low(er)-level concepts/components

- 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.

Notes:

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

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

is an affine transformation.

Notes:

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.

Notes:

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

$$
\begin{array}{rcl}
s'^{(r)} & = & \texttt{AddRoundKey}(s^{(r)}, rk^{(r)}) \\
& = & \texttt{AddRoundKey}\left(\left[\begin{array}{cccc} s^{(r)}_{0,0} & s^{(r)}_{0,1} & s^{(r)}_{0,2} & s^{(r)}_{0,3} \\ s^{(r)}_{1,0} & s^{(r)}_{1,1} & s^{(r)}_{1,2} & s^{(r)}_{1,3} \\ s^{(r)}_{2,0} & s^{(r)}_{2,1} & s^{(r)}_{2,2} & s^{(r)}_{2,3} \\ s^{(r)}_{3,0} & s^{(r)}_{3,1} & s^{(r)}_{3,2} & s^{(r)}_{3,3} \end{array} \right], \left[\begin{array}{cccc} rk^{(r)}_{0,0} & rk^{(r)}_{0,1} & rk^{(r)}_{0,2} & rk^{(r)}_{0,3} \\ rk^{(r)}_{1,0} & rk^{(r)}_{1,1} & rk^{(r)}_{1,2} & rk^{(r)}_{1,3} \\ rk^{(r)}_{2,0} & rk^{(r)}_{2,1} & rk^{(r)}_{2,2} & rk^{(r)}_{2,3} \\ rk^{(r)}_{3,0} & rk^{(r)}_{3,1} & rk^{(r)}_{3,2} & rk^{(r)}_{3,3} \end{array} \right] \right) \\
& = & \left[\begin{array}{cccc}
(s^{(r)}_{0,0} \oplus_{\mathbb{F}_{2^8}} rk^{(r)}_{0,0}) & (s^{(r)}_{0,1} \oplus_{\mathbb{F}_{2^8}} rk^{(r)}_{0,1}) & (s^{(r)}_{0,2} \oplus_{\mathbb{F}_{2^8}} rk^{(r)}_{0,2}) & (s^{(r)}_{0,3} \oplus_{\mathbb{F}_{2^8}} rk^{(r)}_{0,3}) \\
(s^{(r)}_{1,0} \oplus_{\mathbb{F}_{2^8}} rk^{(r)}_{1,0}) & (s^{(r)}_{1,1} \oplus_{\mathbb{F}_{2^8}} rk^{(r)}_{1,1}) & (s^{(r)}_{1,2} \oplus_{\mathbb{F}_{2^8}} rk^{(r)}_{1,2}) & (s^{(r)}_{1,3} \oplus_{\mathbb{F}_{2^8}} rk^{(r)}_{1,3}) \\
(s^{(r)}_{2,0} \oplus_{\mathbb{F}_{2^8}} rk^{(r)}_{2,0}) & (s^{(r)}_{2,1} \oplus_{\mathbb{F}_{2^8}} rk^{(r)}_{2,1}) & (s^{(r)}_{2,2} \oplus_{\mathbb{F}_{2^8}} rk^{(r)}_{2,2}) & (s^{(r)}_{2,3} \oplus_{\mathbb{F}_{2^8}} rk^{(r)}_{2,3}) \\
(s^{(r)}_{3,0} \oplus_{\mathbb{F}_{2^8}} rk^{(r)}_{3,0}) & (s^{(r)}_{3,1} \oplus_{\mathbb{F}_{2^8}} rk^{(r)}_{3,1}) & (s^{(r)}_{3,2} \oplus_{\mathbb{F}_{2^8}} rk^{(r)}_{3,2}) & (s^{(r)}_{3,3} \oplus_{\mathbb{F}_{2^8}} rk^{(r)}_{3,3})
\end{array} \right]
\end{array}
$$

Notes:

Part 1: in theory (7)
High(er)-level concepts/components

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

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

$$
\begin{array}{rcl}
s'^{(r)} & = & \texttt{SubBytes}(s^{(r)}) \\
& = & \texttt{SubBytes}\left(\left[\begin{array}{cccc} s^{(r)}_{0,0} & s^{(r)}_{0,1} & s^{(r)}_{0,2} & s^{(r)}_{0,3} \\ s^{(r)}_{1,0} & s^{(r)}_{1,1} & s^{(r)}_{1,2} & s^{(r)}_{1,3} \\ s^{(r)}_{2,0} & s^{(r)}_{2,1} & s^{(r)}_{2,2} & s^{(r)}_{2,3} \\ s^{(r)}_{3,0} & s^{(r)}_{3,1} & s^{(r)}_{3,2} & s^{(r)}_{3,3} \end{array} \right] \right) \\
& = & \left[\begin{array}{cccc}
\textsc{S-box}(s^{(r)}_{0,0}) & \textsc{S-box}(s^{(r)}_{0,1}) & \textsc{S-box}(s^{(r)}_{0,2}) & \textsc{S-box}(s^{(r)}_{0,3}) \\
\textsc{S-box}(s^{(r)}_{1,0}) & \textsc{S-box}(s^{(r)}_{1,1}) & \textsc{S-box}(s^{(r)}_{1,2}) & \textsc{S-box}(s^{(r)}_{1,3}) \\
\textsc{S-box}(s^{(r)}_{2,0}) & \textsc{S-box}(s^{(r)}_{2,1}) & \textsc{S-box}(s^{(r)}_{2,2}) & \textsc{S-box}(s^{(r)}_{2,3}) \\
\textsc{S-box}(s^{(r)}_{3,0}) & \textsc{S-box}(s^{(r)}_{3,1}) & \textsc{S-box}(s^{(r)}_{3,2}) & \textsc{S-box}(s^{(r)}_{3,3})
\end{array} \right]
\end{array}
$$

Notes:

© Daniel Page (csdsp@bristol.ac.uk)
Applied Cryptology
University of BRISTOL
git # c8178615 @ 2024-04-24

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

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

$$
\begin{array}{rcl}
s'^{(r)} & = & \texttt{ShiftRows}(s^{(r)}) \\
 & = & \texttt{ShiftRows}\left(\left[\begin{array}{cccc} s^{(r)}_{0,0} & s^{(r)}_{0,1} & s^{(r)}_{0,2} & s^{(r)}_{0,3} \\ s^{(r)}_{1,0} & s^{(r)}_{1,1} & s^{(r)}_{1,2} & s^{(r)}_{1,3} \\ s^{(r)}_{2,0} & s^{(r)}_{2,1} & s^{(r)}_{2,2} & s^{(r)}_{2,3} \\ s^{(r)}_{3,0} & s^{(r)}_{3,1} & s^{(r)}_{3,2} & s^{(r)}_{3,3} \end{array}\right]\right) \\
 & = & \left[\begin{array}{cccc} s^{(r)}_{0,0} & s^{(r)}_{0,1} & s^{(r)}_{0,2} & s^{(r)}_{0,3} \\ s^{(r)}_{1,1} & s^{(r)}_{1,2} & s^{(r)}_{1,3} & s^{(r)}_{1,0} \\ s^{(r)}_{2,2} & s^{(r)}_{2,3} & s^{(r)}_{2,0} & s^{(r)}_{2,1} \\ s^{(r)}_{3,3} & s^{(r)}_{3,0} & s^{(r)}_{3,1} & s^{(r)}_{3,2} \end{array}\right]
\end{array}
$$

Notes:

Part 1: in theory (7)
High(er)-level concepts/components

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

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

$$
\begin{array}{rcl}
s'^{(r)} & = & \texttt{MixColumns}(s^{(r)}) \\
 & = & \texttt{MixColumns}\left(\left[\begin{array}{cccc} s^{(r)}_{0,0} & s^{(r)}_{0,1} & s^{(r)}_{0,2} & s^{(r)}_{0,3} \\ s^{(r)}_{1,0} & s^{(r)}_{1,1} & s^{(r)}_{1,2} & s^{(r)}_{1,3} \\ s^{(r)}_{2,0} & s^{(r)}_{2,1} & s^{(r)}_{2,2} & s^{(r)}_{2,3} \\ s^{(r)}_{3,0} & s^{(r)}_{3,1} & s^{(r)}_{3,2} & s^{(r)}_{3,3} \end{array}\right]\right)
\end{array}
$$

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

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

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

Notes:

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

Input: A 128-bit cipher key k
Output: An 11-element sequence rk = AES-128.Enc-KeyExpand(k) ≃ `ExpandRoundKey`(k) of round keys

Generate a sequence of column vectors

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

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**.

Notes:

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 $rk^{(r+1)}$ = AES-128.Enc-KeyEvolve(k) ≃ "`EvolveRoundKey`"($rk^{(r)}, rc^{(r)}$)

1. Compute

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

2. Compute

$$\begin{array}{lclclcl} rk^{(r+1)}_{0,1} & \leftarrow & rk^{(r+1)}_{0,0} \oplus_{\mathbb{F}_{2^8}} rk^{(r)}_{0,1} & \quad & rk^{(r+1)}_{1,1} & \leftarrow & rk^{(r+1)}_{1,0} \oplus_{\mathbb{F}_{2^8}} rk^{(r)}_{1,1} \\ rk^{(r+1)}_{2,1} & \leftarrow & rk^{(r+1)}_{2,0} \oplus_{\mathbb{F}_{2^8}} rk^{(r)}_{2,1} & & rk^{(r+1)}_{3,1} & \leftarrow & rk^{(r+1)}_{3,0} \oplus_{\mathbb{F}_{2^8}} rk^{(r)}_{3,1} \\ \\ rk^{(r+1)}_{0,2} & \leftarrow & rk^{(r+1)}_{0,1} \oplus_{\mathbb{F}_{2^8}} rk^{(r)}_{0,2} & & rk^{(r+1)}_{1,2} & \leftarrow & rk^{(r+1)}_{1,1} \oplus_{\mathbb{F}_{2^8}} rk^{(r)}_{1,2} \\ rk^{(r+1)}_{2,2} & \leftarrow & rk^{(r+1)}_{2,1} \oplus_{\mathbb{F}_{2^8}} rk^{(r)}_{2,2} & & rk^{(r+1)}_{3,2} & \leftarrow & rk^{(r+1)}_{3,1} \oplus_{\mathbb{F}_{2^8}} rk^{(r)}_{3,2} \\ \\ rk^{(r+1)}_{0,3} & \leftarrow & rk^{(r+1)}_{0,2} \oplus_{\mathbb{F}_{2^8}} rk^{(r)}_{0,3} & & rk^{(r+1)}_{1,3} & \leftarrow & rk^{(r+1)}_{1,2} \oplus_{\mathbb{F}_{2^8}} rk^{(r)}_{1,3} \\ rk^{(r+1)}_{2,3} & \leftarrow & rk^{(r+1)}_{2,2} \oplus_{\mathbb{F}_{2^8}} rk^{(r)}_{2,3} & & rk^{(r+1)}_{3,3} & \leftarrow & rk^{(r+1)}_{3,2} \oplus_{\mathbb{F}_{2^8}} rk^{(r)}_{3,3} \end{array}$$

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

Notes:

Part 1: in theory (10)

High(er)-level concepts/components

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 = AES-128.Enc(k, m)

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

Notes:

Part 1: in theory (10)

High(er)-level concepts/components

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

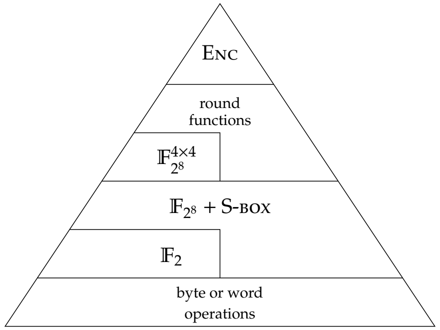
Input: A 128-bit cipher key k, and a 128-bit plaintext message m
Output: A 128-bit ciphertext message c = AES-128.Enc(k, m)

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

Notes:

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, ...

    ⋮

Notes:

Part 2: in practice (2)

$\mathbb{F}_{2^8}$ and $\mathbb{F}_{2^8}^{4\times4}$

- Given $x, y \in \mathbb{F}_{2^8}$,

$$r = x \oplus_{\mathbb{F}_{2^8}} y \equiv x_i + y_i \pmod 2$$
$$\mapsto x_i \oplus y_i$$

for $0 \le i < 8$, and hence

$$x \oplus_{\mathbb{F}_{2^8}} y \equiv x \ominus_{\mathbb{F}_{2^8}} y.$$

Listing

```
uint8_t aes_gf28_add( uint8_t x, uint8_t y ) {
  return x ^ y;
}

uint8_t aes_gf28_sub( uint8_t x, uint8_t y ) {
  return x ^ y;
}
```


Notes:

- Given $x \in \mathbb{F}_{2^8}$, to compute

$$r = \texttt{xtimes}(x) = x \otimes_{\mathbb{F}_{2^8}} \mathbf{x} \equiv \mathbf{x} \times x(\mathbf{x}) \quad (\text{mod}$$

we

1. compute

$$t(\mathbf{x}) = x(\mathbf{x}) \cdot \mathbf{x} = \sum_{i=0}^{i<8} x_i \cdot \mathbf{x}^{i+1},$$

i.e., shift the coefficients, then

2. compute $r(\mathbf{x}) = t(\mathbf{x}) \pmod{p(\mathbf{x})}$ as

$$r(\mathbf{x}) = \begin{cases} t(\mathbf{x}) - p(\mathbf{x}) & \text{if } t_8 = 1 \\ t(\mathbf{x}) & \text{otherwise} \end{cases}$$

i.e., if $t_8 = 1$, we use the fact

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

to reduce $t(\mathbf{x})$ modulo $p(\mathbf{x})$.

Listing

```
uint8_t aes_gf28_mulx( uint8_t x ) {
  if( ( x & 0x80 ) == 0x80 ) {
    return 0x1B ^ ( x << 1 );
  }
  else {
    return        ( x << 1 );
  }
}
```

Notes:

- Given $x, y \in \mathbb{F}_{2^8}$, to compute

$$r = x \otimes_{\mathbb{F}_{2^8}} y$$

we could

1. use polynomial multiplication to compute $x(\mathbf{x}) \cdot y(\mathbf{x})$ *then* reduce this modulo $p(\mathbf{x})$, or
2. use `aes_gf28_mulx` to reduce intermediate results modulo $p(\mathbf{x})$ *during* said polynomial multiplication

in both cases adopting a left-to-right binary (or bit-serial) approach to multiplication.

Listing

```
uint8_t aes_gf28_mul( uint8_t x, uint8_t y ) {
  uint8_t t = 0;

  for( int i =  7; i >= 0; i-- ) {
    t = aes_gf28_mulx( t );

    if( ( y >> i ) & 1 ) {
      t ^= x;
    }
  }

  return t;
}
```

Notes:

Part 2: in practice (4)

$\mathbb{F}_{2^8}$ and $\mathbb{F}_{2^8}^{4\times4}$

- Given $x, y \in \mathbb{F}_{2^8}$, to compute

$$r = x \otimes_{\mathbb{F}_{2^8}} y$$

we could

1. use polynomial multiplication to compute $x(\mathbf{x}) \cdot y(\mathbf{x})$ *then* reduce this modulo $p(\mathbf{x})$, or
2. use `aes_gf28_mulx` to reduce intermediate results modulo $p(\mathbf{x})$ *during* said polynomial multiplication

in both cases adopting a left-to-right binary (or bit-serial) approach to multiplication.

Listing

```
uint8_t aes_gf28_mul( uint8_t x, uint8_t y ) {
  uint16_t t = 0;

  for( int i =  7; i >= 0; i-- ) {
    t <<= 1;

    if( ( y >> i ) & 1 ) {
      t ^= x;
    }
  }

  for( int i = 15; i >= 8; i-- ) {
    if( ( t >> i ) & 1 ) {
      t ^= 0x1B << ( i - 8 );
    }
  }

  return t & 0xFF;
}
```

Notes:

Part 2: in practice (5)

$\mathbb{F}_{2^8}$ and $\mathbb{F}_{2^8}^{4\times4}$

- Given $x \in \mathbb{F}_{2^8}$, to compute

$$r = 1 \oslash_{\mathbb{F}_{2^8}} x$$

we could

1. use the (binary) extended Euclidean algorithm, i.e., compute

$$r(\mathbf{x}) = \mathrm{xgcd}(x(\mathbf{x}), p(\mathbf{x}))$$

or

2. use Lagrange's theorem, i.e.,

$$\begin{array}{lcll} x^q & \equiv & x & \in \mathbb{F}_q \\ x^{q-1} & \equiv & 1 & \in \mathbb{F}_q \\ x^{q-2} & \equiv & x^{-1} & \in \mathbb{F}_q \end{array}$$

where in this case $q = 2^8 = 256$.

Listing

```
uint8_t aes_gf28_inv( uint8_t x ) {
  if( x == 0 ) {
    return 0;
  }
  else {
    uint16_t U = 0x11B, V = x, A = 0, C = 1;

    do {
      while( !( U & 1 ) ) {
        if( A & 1 ) {
          A ^= 0x011B;
        }
        U >>= 1; A >>= 1;
      }

      while( !( V & 1 ) ) {
        if( C & 1 ) {
          C ^= 0x011B;
        }
        V >>= 1; C >>= 1;
      }

      if( U < V ) {
        V ^= U; C ^= A;
      }
      else {
        U ^= V; A ^= C;
      }
    } while( U );

    return C;
  }
}
```

Notes:

- Given $x \in \mathbb{F}_{2^8}$, to compute

$$r = 1 \oslash_{\mathbb{F}_{2^8}} x$$

we could

1. use the (binary) extended Euclidean algorithm, i.e., compute

$$r(\mathbf{x}) = \text{xgcd}(x(\mathbf{x}), p(\mathbf{x}))$$

or

2. use Lagrange's theorem, i.e.,

$$\begin{array}{lclcl} x^q & \equiv & x & \in & \mathbb{F}_q \\ x^{q-1} & \equiv & 1 & \in & \mathbb{F}_q \\ x^{q-2} & \equiv & x^{-1} & \in & \mathbb{F}_q \end{array}$$

where in this case $q = 2^8 = 256$.

Listing

```
uint8_t aes_gf28_inv( uint8_t x ) {
  uint8_t t_0 = aes_gf28_mul(   x,   x ); // x^2
  uint8_t t_1 = aes_gf28_mul( t_0,   x ); // x^3
          t_0 = aes_gf28_mul( t_0, t_0 ); // x^4
          t_1 = aes_gf28_mul( t_1, t_0 ); // x^7
          t_0 = aes_gf28_mul( t_0, t_0 ); // x^8
          t_0 = aes_gf28_mul( t_1, t_0 ); // x^15
          t_0 = aes_gf28_mul( t_0, t_0 ); // x^30
          t_0 = aes_gf28_mul( t_0, t_0 ); // x^60
          t_1 = aes_gf28_mul( t_1, t_0 ); // x^67
          t_0 = aes_gf28_mul( t_0, t_1 ); // x^127
          t_0 = aes_gf28_mul( t_0, t_0 ); // x^254

  return t_0;
}
```

Notes:

- Given $x \in \mathbb{F}_{2^8}$, to compute

$$r = \text{S-BOX}(x)$$

we

1. compute g using `aes_gf28_inv`, then
2. compute f by aligning coefficients in x (plus a constant) such that adding them produces the required result.

Listing

```
uint8_t aes_enc_sbox( uint8_t x ) {
  x = aes_gf28_inv( x );

  x = ( 0x63   ) ^ //   0   1   1   0   0   0   1   1
      ( x      ) ^ // x_7 x_6 x_5 x_4 x_3 x_2 x_1 x_0
      ( x << 1 ) ^ // x_6 x_5 x_4 x_3 x_2 x_1 x_0   0
      ( x << 2 ) ^ // x_5 x_4 x_3 x_2 x_1 x_0   0   0
      ( x << 3 ) ^ // x_4 x_3 x_2 x_1 x_0   0   0   0
      ( x << 4 ) ^ // x_3 x_2 x_1 x_0   0   0   0   0
      ( x >> 7 ) ^ //   0   0   0   0   0   0   0 x_7
      ( x >> 6 ) ^ //   0   0   0   0   0   0 x_7 x_6
      ( x >> 5 ) ^ //   0   0   0   0   0 x_7 x_6 x_5
      ( x >> 4 ) ; //   0   0   0   0 x_7 x_6 x_5 x_4

  return x;
}
```

Notes:

Part 2: in practice (7)
Strategy #1

- 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.

Notes:

Part 2: in practice (8)
Strategy #1

- Selective pre-computation can be effective

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

  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	✓	×

Notes:

Part 2: in practice (9)

Strategy #1

Listing

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

Notes:

Part 2: in practice (9)

Strategy #1

Listing

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

Notes:

© Daniel Page (csdsp@bristol.ac.uk)
Applied Cryptology
University of BRISTOL
git # c8178615 @ 2024-04-24

Listing

```
void aes_enc_rnd_row( uint8_t* s ) {
  AES_ENC_RND_ROW_STEP( 0x1, 0x5, 0x9, 0xD,
                        0xD, 0x1, 0x5, 0x9 );
  AES_ENC_RND_ROW_STEP( 0x2, 0x6, 0xA, 0xE,
                        0xA, 0xE, 0x2, 0x6 );
  AES_ENC_RND_ROW_STEP( 0x3, 0x7, 0xB, 0xF,
                        0x7, 0xB, 0xF, 0x3 );
}
```

Listing

```
#define AES_ENC_RND_ROW_STEP(a,b,c,d,e,f,g,h) {         \
  uint8_t __a1 = s[ a ];                                \
  uint8_t __b1 = s[ b ];                                \
  uint8_t __c1 = s[ c ];                                \
  uint8_t __d1 = s[ d ];                                \
                                                        \
  s[ e ] = __a1;                                        \
  s[ f ] = __b1;                                        \
  s[ g ] = __c1;                                        \
  s[ h ] = __d1;                                        \
}
```


Notes:

Part 2: in practice (9)
Strategy #1

Listing

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

Listing

```
#define AES_ENC_RND_MIX_STEP(a,b,c,d) {                   \
  uint8_t __a1 = s[ a ], __a2 = aes_gf28_mulx( __a1 ); \
  uint8_t __b1 = s[ b ], __b2 = aes_gf28_mulx( __b1 ); \
  uint8_t __c1 = s[ c ], __c2 = aes_gf28_mulx( __c1 ); \
  uint8_t __d1 = s[ d ], __d2 = aes_gf28_mulx( __d1 ); \
                                                        \
  uint8_t __a3 = __a1 ^ __a2;                           \
  uint8_t __b3 = __b1 ^ __b2;                           \
  uint8_t __c3 = __c1 ^ __c2;                           \
  uint8_t __d3 = __d1 ^ __d2;                           \
                                                        \
  s[ a ] = __a2 ^ __b3 ^ __c1 ^ __d1;                   \
  s[ b ] = __a1 ^ __b2 ^ __c3 ^ __d1;                   \
  s[ c ] = __a1 ^ __b1 ^ __c2 ^ __d3;                   \
  s[ d ] = __a3 ^ __b1 ^ __c1 ^ __d2;                   \
}
```


Notes:

Part 2: in practice (10)

Strategy #1

Listing

```c
void aes_enc( uint8_t* c, const uint8_t* m, const uint8_t* k ) {
    uint8_t  s[ 4 * AES_128_NB ];
    uint8_t rk[ 4 * AES_128_NB ];

    memcpy(  s, m, ( 4 * AES_128_NB ) * sizeof( uint8_t ) );
    memcpy( rk, k, ( 4 * AES_128_NB ) * sizeof( uint8_t ) );

  //      1 initial   round
    aes_enc_rnd_key( s, rk );
  // Nr - 1 interated rounds
  for( int r = 1; r < AES_128_NR; r++ ) {
    aes_enc_key_evolve( rk, rk, aes_rcon(          r ) );
    aes_enc_rnd_sub( s     );
    aes_enc_rnd_row( s     );
    aes_enc_rnd_mix( s     );
    aes_enc_rnd_key( s, rk );
  }
  //      1 final     round
    aes_enc_key_evolve( rk, rk, aes_rcon( AES_128_NR ) );
    aes_enc_rnd_sub( s     );
    aes_enc_rnd_row( s     );
    aes_enc_rnd_key( s, rk );

    memcpy(  c, s, ( 4 * AES_128_NB ) * sizeof( uint8_t ) );
}
```

Notes:

Part 2: in practice (11)

Strategy #2

- 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.

Notes:

Part 2: in practice (12)
Strategy #2

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

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

Notes:

Part 2: in practice (12)
Strategy #2

- **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

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

Notes:

Part 2: in practice (12)
Strategy #2

- 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

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

Notes:

Part 2: in practice (12)
Strategy #2

- 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

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

Notes:

- 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.,

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

where

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

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

Notes:

- 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.,

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

where

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

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

Notes:

Part 2: in practice (13)

Strategy #2

Listing

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

Notes:

Part 2: in practice (13)

Strategy #2

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;               \
}
```

Notes:

Part 2: in practice (13)

Strategy #2

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;             \
}
```

Notes:

Part 2: in practice (14)

Strategy #2

Listing

```
void aes_enc( uint8_t* c, const uint8_t* m, const uint8_t* rk ) {
  uint32_t s_0, s_1, s_2, s_3, t_0, t_1, t_2, t_3;

  U8_TO_U32_LE( s_0, m,  0 ); U8_TO_U32_LE( s_1, m,  4 );
  U8_TO_U32_LE( s_2, m,  8 ); U8_TO_U32_LE( s_3, m, 12 );

  uint32_t *rkp = ( uint32_t* )( rk );

  //       1 initial   round
    AES_ENC_RND_INIT();
  // Nr - 1 interated rounds
  for( int i = 1; i < AES_128_NR; i++ ) {
    AES_ENC_RND_ITER();
  }
  //       1 final     round
    AES_ENC_RND_FINI();

  U32_TO_U8_LE( c, s_0,  0 ); U32_TO_U8_LE( c, s_1,  4 );
  U32_TO_U8_LE( c, s_2,  8 ); U32_TO_U8_LE( c, s_3, 12 );
}
```

Notes:

Part 2: in practice (15)

Strategy #3

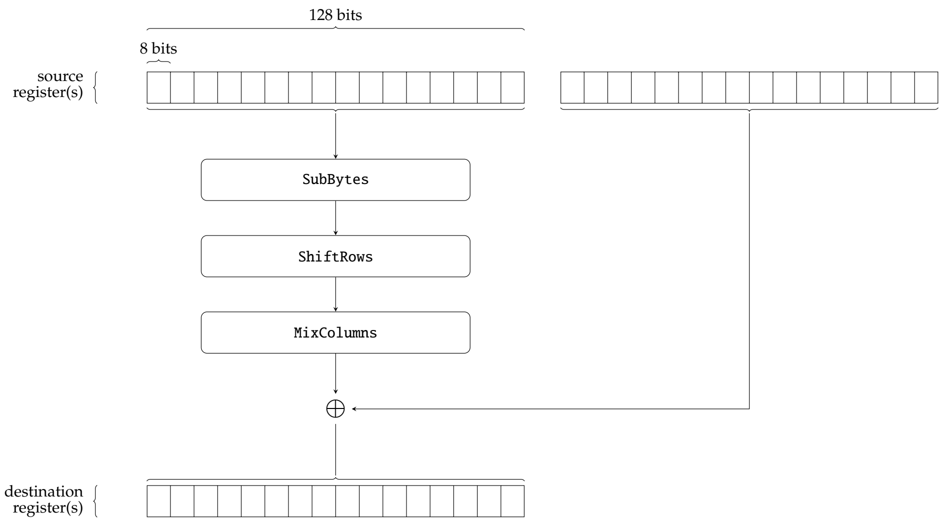
- Strategy #3 [7] (i.e., Intel AES-NI):
  - use hybrid of software (i.e., via ISA) and hardware (i.e., via ISE),
  - adopt a fully-packed representation of state and round key matrices, using 128-bit words, i.e., instances of type `__m128i`,
  - favour use of straight-line (i.e., unrolled) loops,
  -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

Notes:

Part 2: in practice (16)

Strategy #3

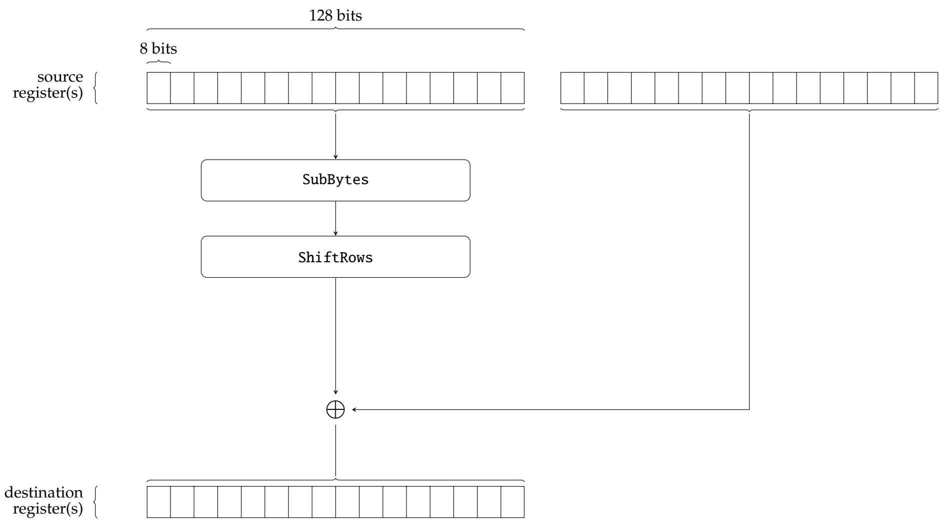
Circuit (`aesenc` [6, Pages 3-54–3-55])



Notes:

Part 2: in practice (16)
Strategy #3

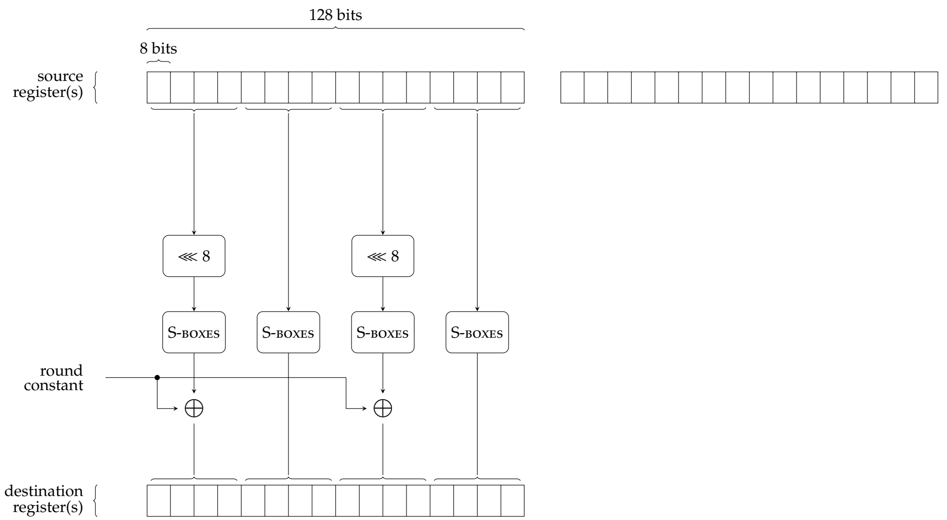
Circuit (`aesenclast` [6, Pages 3-56–3-57])



Notes:

Part 2: in practice (16)
Strategy #3

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



Notes:

Listing

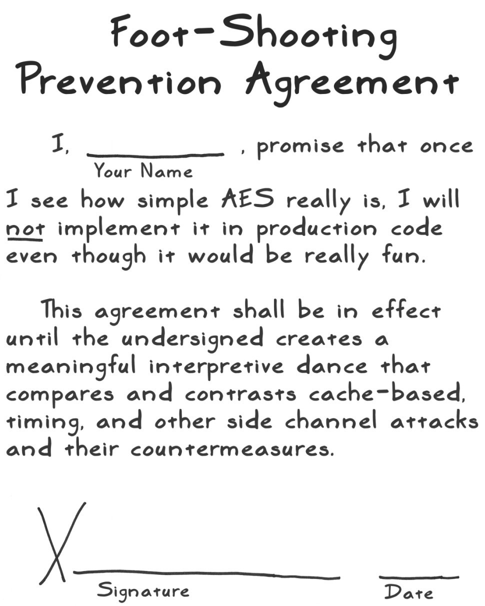
```
void aes_enc( uint8_t* c, const uint8_t* m, const uint8_t* rk ) {
  __m128i s = _mm_load_si128( ( __m128i* )( m ) );
  __m128i* rkp = ( __m128i* )( rk );

  //       1 initial  round
  s =         _mm_xor_si128( s, _mm_load_si128( rkp++ ) );
  // Nr - 1 iterated rounds
  s =      _mm_aesenc_si128( s, _mm_load_si128( rkp++ ) );
  s =      _mm_aesenc_si128( s, _mm_load_si128( rkp++ ) );
  s =      _mm_aesenc_si128( s, _mm_load_si128( rkp++ ) );
  s =      _mm_aesenc_si128( s, _mm_load_si128( rkp++ ) );
  s =      _mm_aesenc_si128( s, _mm_load_si128( rkp++ ) );
  s =      _mm_aesenc_si128( s, _mm_load_si128( rkp++ ) );
  s =      _mm_aesenc_si128( s, _mm_load_si128( rkp++ ) );
  s =      _mm_aesenc_si128( s, _mm_load_si128( rkp++ ) );
  s =      _mm_aesenc_si128( s, _mm_load_si128( rkp++ ) );
  //       1 final    round
  s = _mm_aesenclast_si128( s, _mm_load_si128( rkp++ ) );

  _mm_store_si128( ( __m128i* )( c ), s );
}
```


Notes:

Conclusions



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

Notes:

Conclusions

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

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

  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.

Notes:

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 (update 1). 2023. URL: http://csrc.nist.gov.
- 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.

Notes:

References

[1] *Wikipedia: Advanced Encryption Standard (AES)*. URL: https://en.wikipedia.org/wiki/Advanced_Encryption_Standard (see p. 107).

[2] J. Daemen and V. Rijmen. *The Design of Rijndael*. Springer, 2002 (see pp. 5, 57, 71, 107).

[3] L.R. Knudsen and M.J.B. Robshaw. *The Block Cipher Companion*. Springer, 2011 (see p. 107).

[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. 5).

[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. 5).

[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. 95, 97, 99).

[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. 93).

[8] *Advanced Encryption Standard (AES)*. National Institute of Standards and Technology (NIST) Federal Information Processing Standard (FIPS) 197 (update 1). 2023. URL: http://csrc.nist.gov (see pp. 5, 9, 11, 19, 21, 23, 25, 27, 29, 31, 33, 35, 37, 39, 107).

Notes: