Block ciphers (& AES)

Recall: Symmetric ciphers can be divided into block ciphers & stream ciphers.

* A block cipher encrypts a block of plaintext bits of fixed length l into a block of ciphertext bits & vice versa:

$$E_k(m_1, m_2, \dots, m_l) = (c_1, c_2, \dots, c_l)$$
$$D_k(c_1, c_2, \dots, c_l) = (m_1, m_2, \dots, m_l)$$

Operation modes allow the encryption of plaintexts of arbitrary length.

* A stream cipher generates a keystream $k_1, k_2, \dots$ (a pseudorandom bit sequence) from k, and encrypts the plaintext bits $m_1, m_2, \dots$ by XORing with the keystream. Decryption works similarly.

A block cipher is a keyed family of permutations, which is designed to behave like a pseudorandom permutation (prp). Block ciphers operate on a binary string of fixed length & depend on a key:

$$E: \{0,1\}^n \times \{0,1\}^l \to \{0,1\}^l$$

Today, a block length of $l = 128$ bits and key lengths between $n = 128$ and $n = 256$ bits are used.
↳ key space is large enough to prevent brute-force attacks against uniform random keys

The bijective encryption functions $E_k: \{0,1\}^l \to \{0,1\}^l$ should be indistinguishable from a true random permutation if k is unknown.

The construction of block ciphers is a non-trivial task. Diffusion & confusion (introduced by Claude Shannon, 1949) are important design goals of block ciphers.
↳ Diffusion: small input changes (ex. only one bit) result in large output changes (avalanche effect)
↳ Confusion: complex relationship between ciphertext and key, even if plaintext is known
↳ helps to protect against attacks (ex. EAV, CPA, CCA)

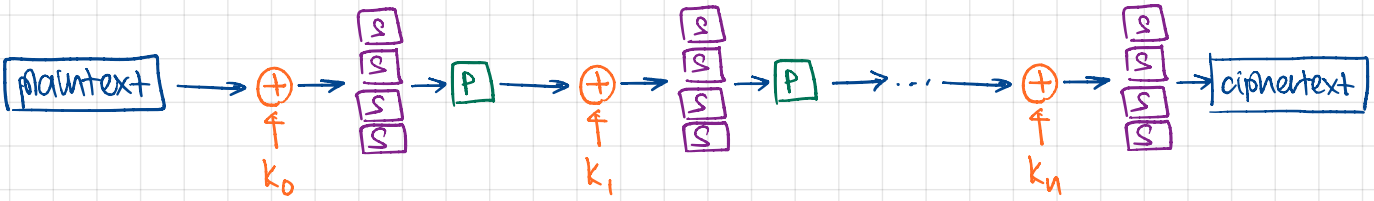
Diffusion can be achieved by linear and affine maps, ex., by XORing with pseudorandom round keys and by matrix operations (linear maps).

Confusion requires non-linear & non-affine maps (so-called S-Boxes). Since non-linear maps can be difficult to describe on full blocks, S-Boxes often operate only on small segments of a block and are applied in parallel, ex., on 8-bit segments of a 128-bit block.

Diffusion and confusion can be achieved by a substitution-permutation network (SPN).
SPN consists of a number of rounds in which a plaintext block is transformed into a ciphertext block. Each round consists of the following operations:

1. Add a round key k_i to the data block. The round key is derived from the encryption key and ensures that the transformation depends on the key.
2. Split the block into smaller segments & apply a non-linear S-Box (substitution) to each of the segments.
3. Apply a bit permutation or, more generally, a linear or affine mixing map to the full data block.

Each operation in a SPN must be invertible so that a unique decryption exists. Typical networks have at least 10 rounds.

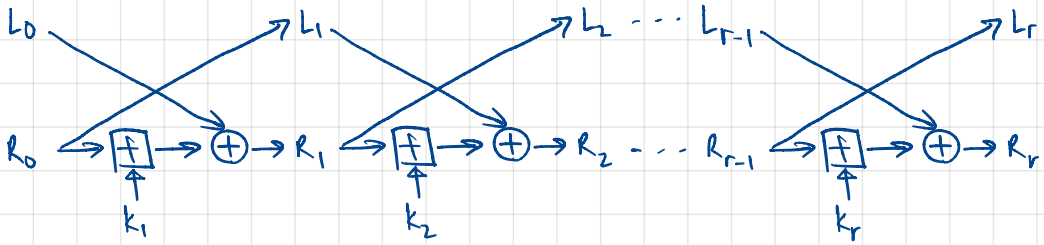


Substitution-permutation network: add a round key k_i, apply the S-Box S and the permutation P in n rounds.

AES is based on this type of network. Another widely-used construction is Feistel networks, which are also based on non-linear S-boxes and linear mixing operations, but split a block into a left and right half. Feistel networks use a round function that depends on a round key and operates on one half of the data block.

*not tested in SE6003 material.

Feistel network with r rounds



Let (L_0, R_0) be plaintext block of length 2t. $(L_i, R_i) = (R_{i-1}, L_{i-1} \oplus f_{k_i}(R_{i-1}))$ for $i = 1, 2, \ldots, r$
Ciphertext after encryption, $E_k(L_0, R_0) = (R_r, L_r)$.

We also define $(R_{i-1}, L_{i-1}) = (L_i, R_i \oplus f_{k_i}(L_i))$ for $i = r, r-1, \ldots, 1$.
Plaintext after decryption, $D_k(R_r, L_r) = (L_0, R_0)$.

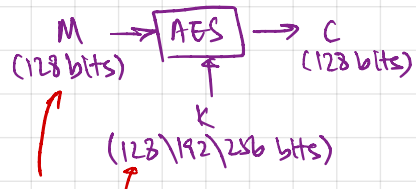
} Encryption & decryption use the same transformation.

The round function f depends on a round key k_i, and f_{k_i} is usually defined by S-boxes and bit permutations, similar to SPN. However, the round function f only operates on one half a block, and due to the Feistel network construction, f does not need to be bijective.

Typical Feistel ciphers have 16 rounds. Block ciphers based on Feistel networks include Data Encryption Standard (DES), Twofish.

NIST adopted Rijndael (designed by J. Daemen & V. Rijmen) as the Advanced Encryption Standard (AES) in 2001; the block cipher is now widely used by security applications & network protocols.

The standardized AES cipher has a block length of 128 bits
& key length of 128, 192 or 256 bits.



Block & key lengths prevent simple brute-force attacks & lookup tables.

The Rijndael encryption function $E_k = \{0,1\}^{128} \to \{0,1\}^{128}$ is given by a SPN.
The number of rounds depend on key size:

- 128-bit key ⇒ 10 rounds
- 192-bit key ⇒ 12 rounds
- 256-bit key ⇒ 14 rounds

} these settings allow for a very efficient operation.

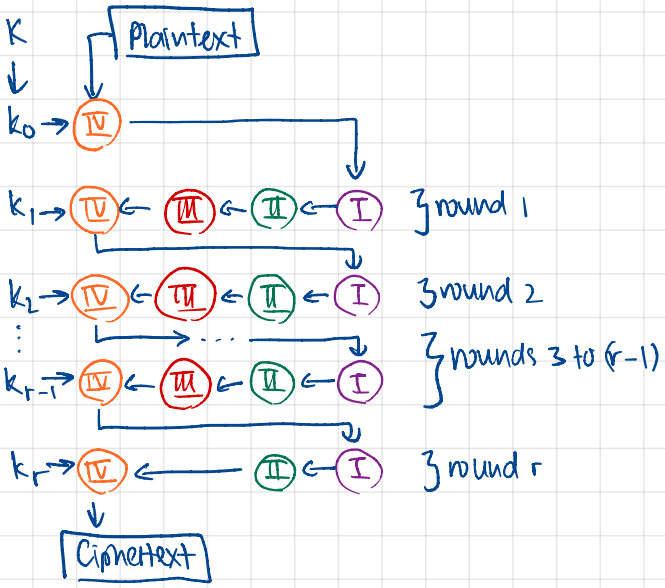
We denote the AES encryption function by $f_k = \{0,1\}^{128} \to \{0,1\}^{128}$.
The 128-bit state is arranged in a 4×4 matrix over $GF(2^8)$ by writing the bytes $P_0, P_1, \ldots, P_{15}$ into the columns of a matrix:

$$\begin{bmatrix} P_0 & P_4 & P_8 & P_{12} \\ P_1 & P_5 & P_9 & P_{13} \\ P_2 & P_6 & P_{10} & P_{14} \\ P_3 & P_7 & P_{11} & P_{15} \end{bmatrix}$$

Each byte is interpreted as an element of the field
$GF(2^8) = GF(2)[x] / (x^8 + x^4 + x^3 + x + 1)$.

```
// High-level description of AES
Rijndael (State, CipherKey) {
    ExpandedKey = KeyExpansion (CipherKey)
    AddRoundKey (State, ExpandedKey[0])
    for (i = 1, i < r; i++) { // r is either 10, 12 or 14
        // Round i
        SubBytes (State)
        ShiftRows (State)
        MixColumns (State)
        AddRoundKey (State, ExpandedKey[i])
    }
    // Final Round
    SubBytes (State)
    ShiftRows (State)
    AddRoundKey (State, ExpandedKey[r])
}
```

Stages in AES:
I. Byte Substitution Layer – provides confusion
II. ShiftRows Layer } diffusion layer
III. MixColumn Layer } └ provides diffusion
IV. Round Key Addition



(I) Byte Substitution Layer

The AES S-Box SubBytes is the only non-affine component of AES.
The S-Box function $S_{RD}: GF(2^8) \to GF(2^8)$ is applied to each byte of the state individually.

$$\begin{bmatrix} P_0 & P_4 & P_8 & P_{12} \\ P_1 & P_5 & P_9 & P_{13} \\ P_2 & P_6 & P_{10} & P_{14} \\ P_3 & P_7 & P_{11} & P_{15} \end{bmatrix} \xrightarrow{\text{SubBytes}} \begin{bmatrix} S_{RD}(P_0) & S_{RD}(4) & S_{RD}(8) & S_{RD}(12) \\ S_{RD}(P_1) & S_{RD}(5) & S_{RD}(9) & S_{RD}(13) \\ S_{RD}(P_2) & S_{RD}(6) & S_{RD}(10) & S_{RD}(14) \\ S_{RD}(P_3) & S_{RD}(7) & S_{RD}(11) & S_{RD}(15) \end{bmatrix}$$

The definition $GF(2^8) = GF(2)[x] / (x^8 + x^4 + x^3 + x + 1)$
gives a $GF(2)$-linear isomorphism between $GF(2^8)$ and $GF(2)^8$, so that elements in $GF(2^8)$ correspond to a binary word of length 8 (one byte).

$$S_{RD}(a) = \begin{cases} Aa^{-1} + b, & b \neq 0 \\ b, & b = 0. \end{cases} \qquad A = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \qquad b = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}$$

Note that implementations of AES do not use the algebraic definition of S_{RD}, but a lookup table instead.

(II) ShiftRows: a bit permutation & rotates the bytes in the 2nd, 3rd & 4th row to the left.

* 1st row is left unchanged
* bytes in
  - 2nd row are rotated by 1 position
  - 3rd row are rotated by 2 positions
  - 4th row are rotated by 3 positions

$$\begin{bmatrix} P_0 & P_4 & P_8 & P_{12} \\ P_1 & P_5 & P_9 & P_{13} \\ P_2 & P_6 & P_{10} & P_{14} \\ P_3 & P_7 & P_{11} & P_{15} \end{bmatrix} \xrightarrow{\text{ShiftRows}} \begin{bmatrix} P_0 & P_4 & P_8 & P_{12} \\ P_5 & P_9 & P_{13} & P_1 \\ P_{10} & P_{14} & P_2 & P_6 \\ P_{15} & P_3 & P_7 & P_{11} \end{bmatrix}$$

(III) MixColumns: transforms the columns of the state by a GF(2^8)-linear map
↳ each column of the state is multiplied by a constant 4×4 matrix M over GF(2^8)
↳ M is invertible.

$$M = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \cdot \begin{bmatrix} P_0 & P_4 & P_8 & P_{12} \\ P_1 & P_5 & P_9 & P_{13} \\ P_2 & P_6 & P_{10} & P_{14} \\ P_3 & P_7 & P_{11} & P_{15} \end{bmatrix} \xrightarrow{\text{MixColumns}} \underbrace{\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix}}_{M} \cdot \begin{bmatrix} P_0 & P_4 & P_8 & P_{12} \\ P_1 & P_5 & P_9 & P_{13} \\ P_2 & P_6 & P_{10} & P_{14} \\ P_3 & P_7 & P_{11} & P_{15} \end{bmatrix}$$

(IV) AddRoundKey: XOR the state and round key k_i.

In the key expansion step, the 128-bit round keys $k_0, k_1, \ldots, k_r$ are derived from the AES key k. The key scheduling is non-linear, which is intended to protect the cipher against related key attacks.

Suppose k is a 128-bit key. Then AES has ten rounds and eleven 128-bit round keys $k_0, k_1, \ldots, k_{10}$ are required. During key expansion, 44 words $W_0, W_1, \ldots, W_{43} \in GF(2^8)^4$ of length 32 bits are computed. The round keys are given by

$$k_i = W_{4i} \| W_{4i+1} \| W_{4i+2} \| W_{4i+3} \text{ for } i = 0, 1, \ldots, 10.$$

Example:
Suppose the cipher key = $\begin{bmatrix} 2b & 28 & ab & 09 \\ 7e & ae & f7 & cf \\ 15 & d2 & 15 & 4f \\ 16 & a6 & 88 & 3c \end{bmatrix}$.

W_0 W_1 W_2 W_3 | W_4 W_5 W_6 W_7 ...

$$\begin{bmatrix} 2b & 28 & ab & 09 \\ 7e & ae & f7 & cf \\ 15 & d2 & 15 & 4f \\ 16 & a6 & 88 & 3c \end{bmatrix} \quad \begin{bmatrix} 89 & 88 & 23 & 2a \\ 84 & 54 & a3 & 6c \\ eb & 2c & 39 & 76 \\ 01 & b1 & 39 & 05 \end{bmatrix} \cdots$$

k_0 k_1

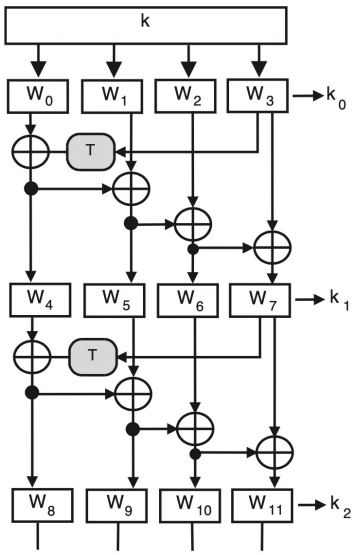


Figure 5.6. The first two rounds of 128-bit AES key scheduling. T maps the word W_{4i-1} to $\mathbf{S}(sh(W_{4i-1})) \oplus (RC_i, 0, 0, 0)$. This is basically a byte-wise SubBytes operation, but involves an additional rotation and a translation by a round constant.

$$W_4 = W_0 \oplus S_{RD}(\text{RotWord}(W_3)) \oplus \text{Rcon}(4)$$
$$= \begin{bmatrix} 2b \\ 7e \\ 15 \\ 16 \end{bmatrix} \oplus \begin{bmatrix} 89 \\ 84 \\ eb \\ 01 \end{bmatrix} \oplus \begin{bmatrix} 01 \\ 00 \\ 00 \\ 00 \end{bmatrix} = \begin{bmatrix} a0 \\ fa \\ fe \\ 17 \end{bmatrix}$$

$$\text{RotWord}(W_3) = \text{RotWord}\left(\begin{bmatrix} 09 \\ cf \\ 4f \\ 3c \end{bmatrix}\right) = \begin{bmatrix} cf \\ 4f \\ 3c \\ 09 \end{bmatrix}$$

$$\text{Rcon} = \begin{bmatrix} 01 & 02 & 04 & 08 & 10 & 20 & 40 & 80 & 1b & 36 \\ 00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 \\ 00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 \\ 00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 & 00 \end{bmatrix}$$
(4) (8) (12) (16) (20) (28) (32) (36) (40) (44)

$$W_5 = W_1 \oplus W_4$$
$$W_6 = W_2 \oplus W_5$$
$$W_7 = W_3 \oplus W_6$$
$$W_8 = W_4 \oplus S_{RD}(\text{RotWord}(W_7)) \oplus \text{Rcon}(8)$$
$$= \begin{bmatrix} a0 \\ fa \\ fe \\ 17 \end{bmatrix} \oplus \begin{bmatrix} 50 \\ 38 \\ 6b \\ e5 \end{bmatrix} \oplus \begin{bmatrix} 02 \\ 00 \\ 00 \\ 00 \end{bmatrix} = \begin{bmatrix} f2 \\ c2 \\ 95 \\ f2 \end{bmatrix}$$

Words in positions that are a multiple of 4 ($W_4, W_8, \ldots, W_{40}$) are calculated by:
(a) applying the RotWord and SubBytes transformation to the previous word W_{i-1}.
(b) XOR this result to the word 4 positions earlier W_{i-4}, plus a round constant Rcon.
The remaining 32-bit words W_i are calculated by XORing the previous word W_{i-1} with the word 4 positions earlier W_{i-4}.