

Cryptography

Symmetric Encryption

Prof. Dr. Heiko Knospe

TH Köln – University of Applied Sciences

March 31, 2024

Block Ciphers

Symmetric ciphers can be divided into *block ciphers* and *stream ciphers*. A *block cipher* encrypts a block of plaintext bits of fixed length l into a block of ciphertext bits, and vice versa:

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

$$E_k^{-1}(c_1, c_2, \dots, c_l) = (m_1, m_2, \dots, m_l)$$

Operations modes allow the encryption of plaintexts of arbitrary length.

Stream Ciphers

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

$$(c_1, c_2, \dots) = (m_1, m_2, \dots) \oplus (k_1, k_2, \dots) = (m_1 \oplus k_1, m_2 \oplus k_2, \dots)$$
$$(m_1, m_2, \dots) = (c_1, c_2, \dots) \oplus (k_1, k_2, \dots) = (m_1 \oplus k_1 \oplus k_1, m_2 \oplus k_2 \oplus k_2, \dots)$$

CTR Mode

The CTR mode transforms a block cipher into a stream cipher. The keystream is given by encrypting a randomly chosen counter. By incrementing the counter the required number of keystream blocks is generated.

$$\begin{aligned}ctr &\xleftarrow{\$} \{0, 1\}^I \\(k_1, \dots, k_l) &= E_k(ctr + 1) \\(k_{l+1}, \dots, k_{2l}) &= E_k(ctr + 2) \\&\dots\end{aligned}$$

Native Stream Ciphers

Stream ciphers have been used a lot in the past and they are often very efficient. Examples include the A5 ciphers (for GSM encryption) and RC4 (often used for network data encryption). However, these ciphers have cryptographic weaknesses (they are not pseudorandom generators) and they are deprecated now.

A new generation of stream ciphers is under investigation, but new ciphers are not yet standardized. A promising candidate is the *Salsa20 family of stream ciphers*, in particular the high-speed cipher *ChaCha20* (see [RFC 7539](#)), which is already used in the network security protocols TLS, SSH and IPsec as an alternative to AES.

Block Ciphers

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 and depend on a key:

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

Today, a block length of $l = 128$ bits and key lengths between $n = 128$ and $n = 256$ bits are used.

Design of Block Ciphers

The bijective encryption functions

$$E_k : \{0, 1\}^l \rightarrow \{0, 1\}^l$$

should be indistinguishable from a true random permutation if k is unknown. The construction of a secure block cipher (*prp*) is a non-trivial task! *Diffusion* and *confusion* are important design goals of block ciphers.

- Diffusion: Small input changes, e.g., only one bit, result in large output changes (avalanche effect). Plaintext patterns are diffused in the ciphertext.
- Confusion: Complex and non-linear relationship between ciphertext and key, even if plaintext is known. This helps to protect against cryptanalysis, e.g. with linear approximations, attacks where plaintext-ciphertext pairs are known, e.g., CPA, CCA, and also protects intermediate round keys.

Diffusion and Confusion

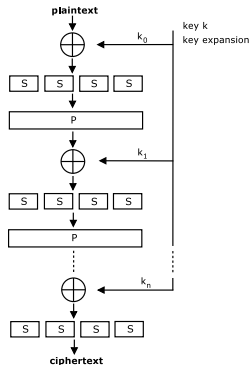
Diffusion can be achieved by linear and affine maps, e.g., by (bit or byte) permutations, XOR and matrix operations. Such operations can have good diffusion properties and are very efficient. However, they can be easily reconstructed by linear algebra.

Confusion requires *nonlinear* and *non-affine* maps (so-called S-Boxes). Since nonlinear maps are difficult to describe on full blocks, S-Boxes usually operate only on small segments of a block and are applied in parallel, e.g., on 8-bit segments of a 128-bit block.

Modern block ciphers combine diffusion and confusion operations to obtain a key-dependent non-linear transformation.

Substitution-Permutation Networks

In practice, *Substitution-Permutation Networks* (SPN, see figure below) and *Feistel networks* are used to construct block ciphers.



Plaintext is transformed into ciphertext in several invertible rounds. A round key k_i is added, P is an affine or linear permutation box and S is a non-linear S-Box.

AES Encryption

The block cipher *Rijndael* has been adopted as *Advanced Encryption Standard (AES)* and the cipher is widely used today. The standardized AES cipher has a block length of 128 bits and supports 128-, 192- and 256-bit key lengths. The Rijndael encryption function

$$E_k : \{0, 1\}^{128} \rightarrow \{0, 1\}^{128}$$

is given by a SPN. The 128-bit *state* is arranged in a 4×4 matrix over $GF(2^8)$ by writing the bytes p_0, p_1, \dots, p_{16} into the columns.

$$\begin{pmatrix} p_0 & p_4 & p_8 & p_{12} \\ p_1 & p_5 & p_9 & p_{13} \\ p_2 & p_6 & p_{10} & p_{14} \\ p_3 & p_7 & p_{11} & p_{15} \end{pmatrix}$$

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)
{
    KeyExpansion(CipherKey, ExpandedKey) // Derive the round keys
    AddRoundKey(State, ExpandedKey[0])
    for(i = 1; i < Nr ; i++) {          // Nr 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[Nr])
}
```

AES S-Box

The AES S-Box `SubBytes` is the only non-affine component of AES.

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

$$\begin{pmatrix} p_0 & p_4 & p_8 & p_{12} \\ p_1 & p_5 & p_9 & p_{13} \\ p_2 & p_6 & p_{10} & p_{14} \\ p_3 & p_7 & p_{11} & p_{15} \end{pmatrix} \xrightarrow{\text{SubBytes}} \begin{pmatrix} S_{RD}(p_0) & S_{RD}(p_4) & S_{RD}(p_8) & S_{RD}(p_{12}) \\ S_{RD}(p_1) & S_{RD}(p_5) & S_{RD}(p_9) & S_{RD}(p_{13}) \\ S_{RD}(p_2) & S_{RD}(p_6) & S_{RD}(p_{10}) & S_{RD}(p_{14}) \\ S_{RD}(p_3) & S_{RD}(p_7) & S_{RD}(p_{11}) & S_{RD}(p_{15}) \end{pmatrix}$$

S-Box Function S_{RD}

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 $GF(2)^8$. The invertible S-Box function is defined by a multiplicative inversion in $GF(2^8)^*$ (which is highly nonlinear), followed by an affine map:

$$S_{RD} : GF(2^8) \rightarrow GF(2^8), S_{RD}(a) = \begin{cases} Aa^{-1} + b & \text{for } a \neq 0 \\ b & \text{for } a = 0 \end{cases}$$

$$A = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{pmatrix} \quad b = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{pmatrix}$$

ShiftRows

ShiftRows is a bit-permutation and rotates the bytes in the second, third and fourth row to the left. The first row is left unchanged, the bytes in second row are rotated by one position, bytes in the third row are rotated by two positions and bytes in the fourth row are rotated by three positions. Obviously, *ShiftRows* is invertible.

$$\begin{pmatrix} p_0 & p_4 & p_8 & p_{12} \\ p_1 & p_5 & p_9 & p_{13} \\ p_2 & p_6 & p_{10} & p_{14} \\ p_3 & p_7 & p_{11} & p_{15} \end{pmatrix} \xrightarrow{\text{ShiftRows}} \begin{pmatrix} p_0 & p_4 & p_8 & p_{12} \\ p_5 & p_9 & p_{13} & p_1 \\ p_{10} & p_{14} & p_2 & p_6 \\ p_{15} & p_3 & p_7 & p_{11} \end{pmatrix}$$

MixColumns

MixColumns transforms the columns of the state by a $GF(2^8)$ -linear map. The product of a constant 4×4 matrix M over $GF(2^8)$ with the current state matrix defines the new state, i.e., each column of the state is multiplied with M and gives the updated column. The matrix M is invertible over $GF(2^8)$.

$$\begin{pmatrix} p_0 & p_4 & p_8 & p_{12} \\ p_1 & p_5 & p_9 & p_{13} \\ p_2 & p_6 & p_{10} & p_{14} \\ p_3 & p_7 & p_{11} & p_{15} \end{pmatrix} \xrightarrow{\text{MixColumns}} \underbrace{\begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix}}_M \cdot \begin{pmatrix} p_0 & p_4 & p_8 & p_{12} \\ p_1 & p_5 & p_9 & p_{13} \\ p_2 & p_6 & p_{10} & p_{14} \\ p_3 & p_7 & p_{11} & p_{15} \end{pmatrix}$$

Key Scheduling

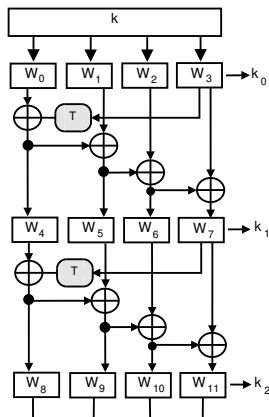
In the key expansion step, the 128-bit round keys k_0, k_1, \dots, k_r are derived from the AES key k . The round keys ensure that the transformation depends on the key. Note that confusion in the cipher protects the round keys against cryptanalysis.

The AES key scheduling is also *nonlinear*, which is intended to protect the cipher against *related key attacks*.

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

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

Key Scheduling for 128-bit AES keys



The first two rounds of 128-bit AES key scheduling (round keys k_0, k_1, k_2). T is defined by $T(W_{4i-1}) = \mathbf{S}(sh(W_{4i-1})) \oplus (RC_i, 0, 0, 0)$, where sh rotates the bytes to the left, $\mathbf{S} = S_{RD} \times S_{RD} \times S_{RD} \times S_{RD}$ and RC_i is a round constant.