Cryptography 101: From Theory to Practice

Chapter 9 – Symmetric Encryption

Rolf Oppliger

March 8, 2022

Terms of Use

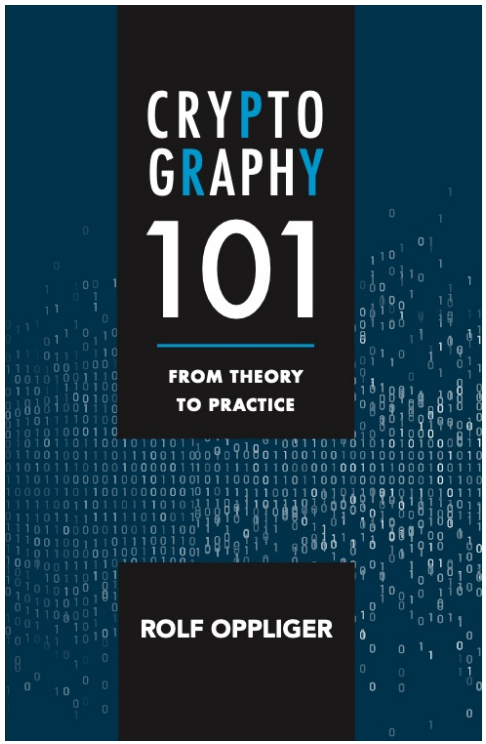
- This work is published with a CC BY-ND 4.0 license ()
  - CC = Creative Commons ()
  - BY = Attribution ()
  - ND = No Derivatives ()

whoami

`rolf-oppliger.ch`
`rolf-oppliger.com`

- Swiss National Cyber Security Centre NCSC (scientific employee)
- eSECURITY Technologies Rolf Oppliger (founder and owner)
- University of Zurich (adjunct professor)
- Artech House (author and series editor for information security and privacy)

Reference Book



ISBN 978-1-63081-846-3

`https://books.esecurity.ch/crypto101.html`

Challenge Me

Outline

9. Symmetric Encryption

9. Symmetric Encryption

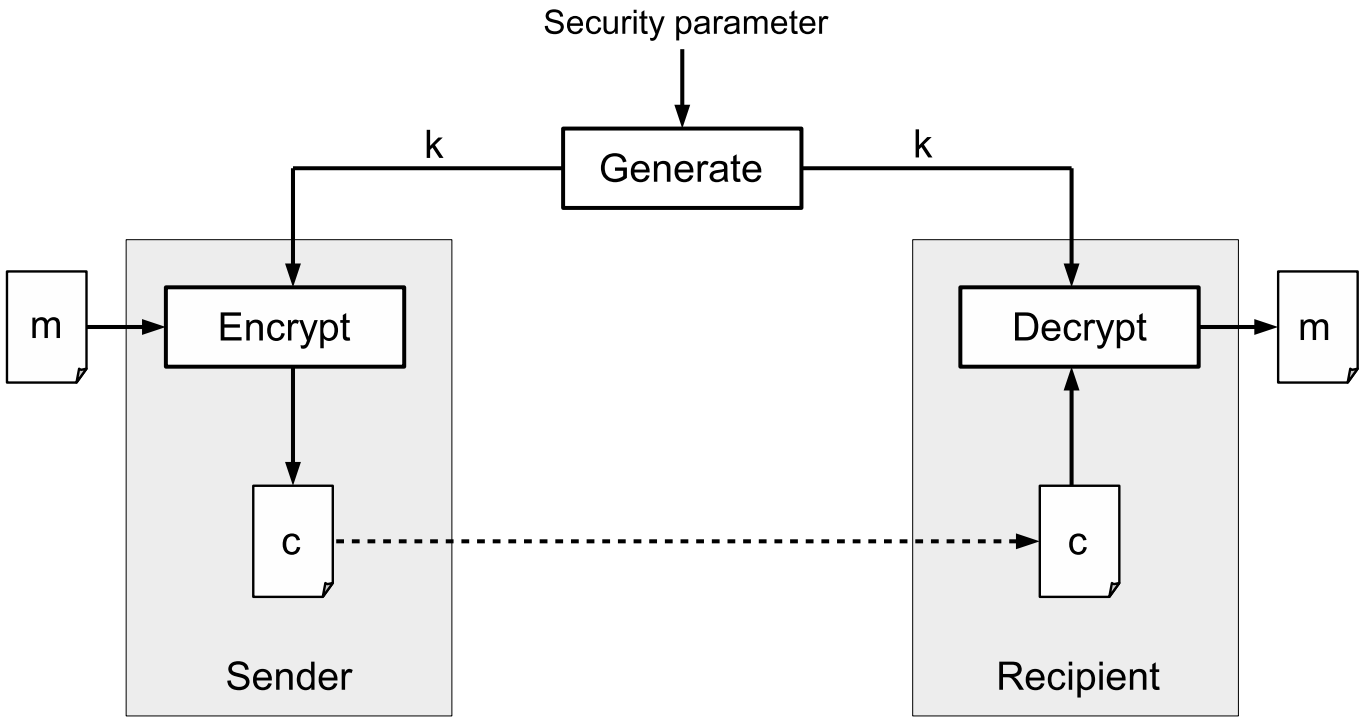
9. Symmetric Encryption

9.1 Introduction

- According to Definition 2.9, a **symmetric encryption system** (or **cipher**) is a pair (E, D) of families of efficiently computable functions
  - $E : \mathcal{K} \times \mathcal{M} \to \mathcal{C}$ refers to the family $\{E_k : k \in \mathcal{K}\}$ of *encryption functions* $E_k : \mathcal{M} \to \mathcal{C}$
  - $D : \mathcal{K} \times \mathcal{C} \to \mathcal{M}$ refers to the family $\{D_k : k \in \mathcal{K}\}$ of *decryption functions* $D_k : \mathcal{C} \to \mathcal{M}$
- For every message $m \in \mathcal{M}$ and key $k \in \mathcal{K}$, the functions D_k and E_k must be inverse to each other, i.e., $D_k(E_k(m)) = m$

9. Symmetric Encryption

9.1 Introduction



9. Symmetric Encryption

9.1 Introduction

- If $\mathcal{M}$ and $\mathcal{C}$ are the same, then it usually doesn't matter whether one encrypts first and then decrypts or decrypts first and then encrypts, i.e., $D_k(E_k(m)) = E_k(D_k(m)) = m$
- A system is **commutative** if a message that is encrypted multiple times can be decrypted in arbitrary order
- For example, if m is encrypted twice with a commutative encryption system (and keys k_1 and k_2), i.e., $c = E_{k_2}(E_{k_1}(m))$, then $D_{k_2}(D_{k_1}(c)) = D_{k_1}(D_{k_2}(c)) = m$

9. Symmetric Encryption

9.1 Introduction

- In order to evaluate a particular (symmetric) encryption system, one needs well-defined criteria
- Shannon's criteria
  - Amount of secrecy
  - Size of key
  - Complexity of enciphering and deciphering operations
  - Propagation of errors
  - Expansion of messages
- This list is not comprehensive, and other (or rather complementary) evaluation criteria may be important in a particular environment or application setting

9. Symmetric Encryption

9.1 Introduction

- Every practically relevant symmetric encryption system processes plaintext messages unit by unit
- A unit may be either a bit or a block of bits (e.g., one or several bytes)
- The i-th ciphertext unit depends on the i-th plaintext unit, the key, and possibly some internal state
- Consequently, there are two types of ciphers
  - **Block ciphers** have no internal state
  - **Stream ciphers** have internal state

9. Symmetric Encryption

9.1 Introduction

- Furthermore, there are two subtypes of stream ciphers
  - In a **synchronous** (or **additive**) stream cipher, the next state does not depend on the previously generated ciphertext units
  - In a **nonsynchronous** (or **self-synchronizing**) stream cipher, the next state also depends on some (or all) previously generated ciphertext units
- Synchronous (additive) stream ciphers are more widely used in the field
- A synchronous (additive) stream cipher is essentially a PRG

9. Symmetric Encryption

9.1 Introduction

- To argue about the security of a symmetric encryption system one has to specify
  - the adversary and the attacks he or she is able to mount
  - the task he or she is required to solve to be successful
- Attacks
  - Ciphertext-only attack (are always possible)
  - Known-plaintext attack
  - (Adaptive) chosen-plaintext attack (CPA / CPA2)
  - (Adaptive) chosen-ciphertext attack (CCA / CCA2)
- In most situations, brute-force or exhaustive key search attacks are possible

9. Symmetric Encryption

9.2 Historical Perspective

- Every cipher employs one or several alphabet(s) to form the plaintext message, ciphertext, and key spaces
- For example, $\Sigma = \{A, \ldots, Z\} \cong \mathbb{Z}_{26} = \{0, 1, \ldots, 25\}$)
- If $\mathcal{M} = \mathcal{C} = \mathcal{K} = \mathbb{Z}_{26}$, then an **additive cipher** can be defined as

$$E_k: \quad \begin{array}{l} \mathcal{M} \longrightarrow \mathcal{C} \\ m \longmapsto (m + k) \bmod 26 \end{array} \qquad D_k: \quad \begin{array}{l} \mathcal{C} \longrightarrow \mathcal{M} \\ c \longmapsto (c - k) \bmod 26 \end{array}$$

- In this setting, the **Caesar cipher** is an additive cipher with $k = 3$

9. Symmetric Encryption

9.2 Historical Perspective

- Similar to the additive cipher, one can define a **multiplicative cipher** or combine an additive and a multiplicative cipher in an **affine cipher**
- In an affine cipher, $\mathcal{K}$ consists of all pairs $(a, b) \in \mathbb{Z}_{26}^2$ with $gcd(a, 26) = 1$ (i.e., $|\mathcal{K}| = \phi(26) \cdot 26 = 312$)
- An affine cipher can be defined as

$$E_{(a,b)}: \quad \begin{array}{l} \mathcal{M} \longrightarrow \mathcal{C} \\ m \longmapsto (am + b) \bmod 26 \end{array} \qquad D_{(a,b)}: \quad \begin{array}{l} \mathcal{C} \longrightarrow \mathcal{M} \\ c \longmapsto (a^{-1}(c - b)) \bmod 26 \end{array}$$

- Note that the multiplicative inverse element of a in $\mathbb{Z}_{26}$, i.e., $a^{-1} \bmod 26$, is needed to decrypt c

9. Symmetric Encryption

9.2 Historical Perspective

- $\Sigma = \{A, \ldots, Z\} \cong \mathbb{Z}_{26}$ is a good choice for human beings
- If computer systems are used for encryption and decryption, then it is advantageous and more appropriate to use $\Sigma = \mathbb{Z}_2 = \{0, 1\} \cong \mathbb{F}_2$
- The plaintext message and ciphertext spaces are set to $\{0, 1\}^*$, whereas the key space is set to $\{0, 1\}^l$ for a reasonable key length l
- In practice, l is often set to 128 or 256 bits

9. Symmetric Encryption

9.2 Historical Perspective

- Additive, multiplicative, and affine ciphers are **monoalphabetic substitution ciphers**
- Each letter of the plaintext alphabet is replaced by another (always the same) letter of the ciphertext alphabet
- A monoalphabetic substitution cipher can be thought of a permutation of the letters that form the (plaintext and ciphertext) alphabet Σ
- For example, there are $|\Sigma|! = 26! > 4 \cdot 10^{26}$ permutations of the 26 letters of the Latin alphabet

9. Symmetric Encryption

9.2 Historical Perspective

- The key space of such a monoalphabetic substitution cipher is huge (this defeats exhaustive key search)
- But a monoalphabetic substitution cipher cannot disguise the frequency distributions of individual (groups of) letters
- It is therefore possible to decrypt a ciphertext using statistical techniques
- An early attempt to defeat frequency analysis attacks was to disguise plaintext letter frequencies by homophony
- In a **homophonic substitution cipher**, a plaintext letter is replaced by a ciphertext letter from a group

9. Symmetric Encryption

9.2 Historical Perspective

- Alternatively, a **polyalphabetic substitution ciphers** can flatten the frequency distribution of ciphertext letters by using multiple ciphertext alphabets in a cyclic way
- The most important examples are the **Vigenère cipher** used in the Middle Ages and the **Enigma** machine used by the Germans in World War II
- From today's perspective, all substitution ciphers are insecure and can be cryptanalyzed
- Perfectly or computationally secure ciphers are needed instead

9. Symmetric Encryption

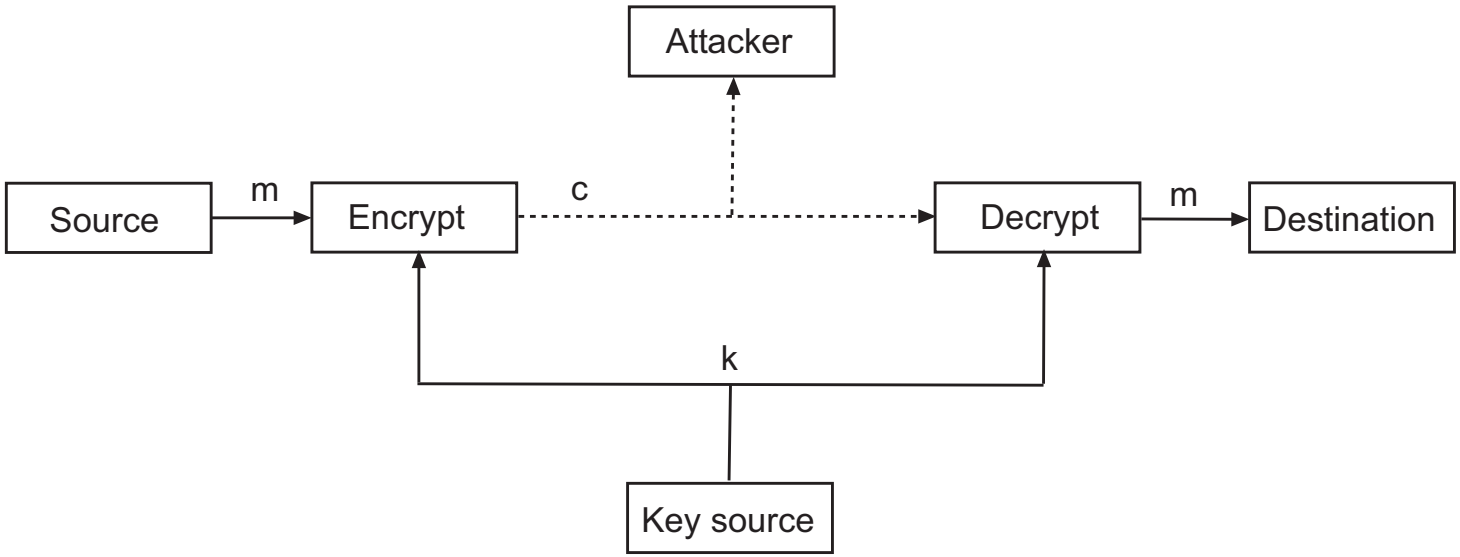
9.3 Perfectly Secure Encryption

- The field of perfectly (i.e., information-theoretically) secure encryption was pioneered by Shannon in the late 1940s
- The aim was to come up with an encryption system that is secure in the sense that it is impossible for an adversary to derive any information about a plaintext message from a given ciphertext
- This must be true even if the adversary has the best available computer technology, and even if he or she is not limited in terms of computational resources (e.g., time and memory)
- Such an absolute notion of security exists, but it is usually too expensive to achieve in practice

9. Symmetric Encryption

9.3 Perfectly Secure Encryption

Shannon's model of a symmetric encryption system



9. Symmetric Encryption

9.3 Perfectly Secure Encryption

- In Shannon's model, let $m_0, m_1 \in \mathcal{M}$ be plaintext messages with $|m_0| = |m_1|$, and $c \in \mathcal{C}$ be the encryption of either m_0 or m_1
- If the encryption is perfectly secure, then the probability that c is the encryption of m_0 must be equal to the probability that c is the encryption of m_1:

$$\Pr_{k \xleftarrow{r} \mathcal{K}}[E_k(m_0) = c] = \Pr_{k \xleftarrow{r} \mathcal{K}}[E_k(m_1) = c]$$

- If this is true for all $m_0, m_1 \in \mathcal{M}$, $c \in \mathcal{C}$, and $k \xleftarrow{r} \mathcal{K}$, then c can yield no information about the plaintext message

9. Symmetric Encryption

9.3 Perfectly Secure Encryption

- Alternatively, an encryption (process) that takes place in a symmetric encryption system (E, D) over $\mathcal{M}$, $\mathcal{C}$, and $\mathcal{K}$ can be seen as a discrete random experiment
- M and K are then real-valued random variables distributed according to the probability distributions $P_M : \mathcal{M} \to \mathbb{R}^+$ and $P_K : \mathcal{K} \to \mathbb{R}^+$
- Note that P_M typically depends on the language in use, whereas P_K is often uniformly distributed over all possible keys (i.e., all keys are equally probable)
- In either case, it is reasonable to assume that M and K are independent from each other

9. Symmetric Encryption

9.3 Perfectly Secure Encryption

- In addition to M and K, there is a third random variable C distributed according to $P_C : \mathcal{C} \rightarrow \mathbb{R}^+$
- This random variable stands for the ciphertext
- The probability distribution P_C completely depends on P_M and P_K
- The random variable C is the one that can be observed by the (passive) adversary and from which he or she may try to derive information about M or K

9. Symmetric Encryption

9.3 Perfectly Secure Encryption

- If the two random variables M and C are independent from each other, then C yields no information about M
- This suggests that the a priori probability distribution P_M and the a posteriori probability distribution $P_{M|C}$ must be equal (for an encryption to be perfectly secure)

Definition 9.1 (Perfectly secure symmetric encryption system)

A symmetric encryption system (E, D) over $\mathcal{M}$, $\mathcal{C}$, and $\mathcal{K}$ is perfectly secure if $P_M = P_{M|C}$

9. Symmetric Encryption

9.3 Perfectly Secure Encryption

- Alternatively, the notion of perfect security (secrecy) can be defined information-theoretically using the notion of entropy

Definition 9.2 (Perfectly secure symmetric encryption system)

A symmetric encryption system (E, D) over $\mathcal{M}$, $\mathcal{C}$, and $\mathcal{K}$ is perfectly secure if $H(M|C) = H(M)$ for every probability distribution P_M

9. Symmetric Encryption

9.3 Perfectly Secure Encryption

- Shannon showed for (nonrandomized) symmetric encryption systems that a necessary but usually not sufficient condition for such a system to be perfectly secure is that the entropy of K is at least as large as the entropy of M
- This implies that the secret key must be at least as long as the plaintext

Theorem (Shannon)

In a perfectly secure symmetric encryption system $H(K) \geq H(M)$

9. Symmetric Encryption

9.3 Perfectly Secure Encryption

- The prime example of a perfectly secure cipher is the **one-time pad (OTP)** credited to Gilbert S. Vernam (aka **Vernam cipher**)
- It consists of a randomly generated stream of key bits $k = k_1, k_2, k_3, \ldots$ shared between the sender and the recipient
- To encrypt plaintext message $m = m_1, m_2, \ldots, m_n$, the sender adds each bit m_i $(1 \leq i \leq n)$ modulo 2 with k_i:

$$c_i = m_i \oplus k_i \text{ for } i = 1, \ldots, n$$

- The ciphertext $c = c_1, c_2, c_3, \ldots, c_n$ is sent to the recipient

9. Symmetric Encryption

9.3 Perfectly Secure Encryption

- The recipient can recover the plaintext by adding each ciphertext bit c_i modulo 2 with the respective key bit k_i:

$$c_i \oplus k_i = (m_i \oplus k_i) \oplus k_i = m_i \oplus (k_i \oplus k_i) = m_i \oplus 0 = m_i$$

Theorem (One-time pad)

The one-time pad provides perfect secrecy

The proof only applies if the key is truly random and used only once

9. Symmetric Encryption

9.3 Perfectly Secure Encryption

- The OTP provides perfect security in terms of secrecy, but it neither provides integrity nor authenticity
- It is possible to modify a known ciphertext so that it has a well-defined effect on the underlying plaintext message
- This means that the one-time pad is highly malleable
- There are situations in which malleability is a desired property (e.g., plausible deniability)
- In most situations, however, nonmalleability is the desired property and the primary design goal

9. Symmetric Encryption

9.4 Computationally Secure Encryption

- In a perfectly secure encryption system, no information about a plaintext message leaks from a ciphertext
- In a computationally secure encryption system, one only requires that the amount of information that may leak and can therefore be extracted is negligible
- This leads to the notion of **semantic security**
- An encryption system is **semantically secure** if it is computationally infeasible to derive significant information about a plaintext message from a ciphertext

9. Symmetric Encryption

9.4 Computationally Secure Encryption

- It is generally difficult to prove semantic security
- But it is equivalent to **ciphertext indistinguishability under a CPA (IND-CPA)**
- This notion of security can be explained and formally defined in the security or (in)distinguishability game
- A symmetric encryption system is computationally secure, if it is semantically secure and hence provides IND-CPA

9. Symmetric Encryption

9.4 Computationally Secure Encryption

- Let (E, D) over $\mathcal{M}$, $\mathcal{C}$, and $\mathcal{K}$ be a symmetric encryption system for which IND-CPA needs to be shown
  - The adversary chooses a pair of equally long plaintext messages m_0 and m_1 and sends them to the challenger
  - The challenger randomly selects a key $k \in_R \mathcal{K}$ and a bit $b \in_R \{0, 1\}$, and sends $c = E_k(m_b)$ to the adversary
  - After q such queries, the adversary has to choose and output b' that is 0 (if c is the encryption of m_0) or 1 (if c is the encryption of m_1)
- The adversary is successful if the probability that $b' = b$ is significantly better than guessing, i.e., $\Pr[b' = b] = \frac{1}{2} + \epsilon$ for some nonnegligible ϵ

9. Symmetric Encryption

9.5 Stream Ciphers

- Stream ciphers have played and continue to play an important role in (the history of) cryptography
- The i-th ciphertext unit thus depends on the i-th plaintext unit, the key, and some internal state
- There are synchronous (additive) and nonsynchronous (self-synchronizing) stream ciphers
- Most stream ciphers in use today are synchronous (additive) and conceptually similar to the OTP ("pseudo OTP")

9. Symmetric Encryption

9.5 Stream Ciphers

- Let $\Sigma = \mathbb{Z}_2 = \{0, 1\}$, $\mathcal{M} = \mathcal{C} = \Sigma^*$, and $\mathcal{K} = \Sigma^n$ for key length n
- To encrypt l-bit plaintext message $m = m_1, \ldots, m_l \in \mathcal{M}$ using an additive stream cipher, an n-bit key $k \in \mathcal{K}$ must be expanded into a stream of l key bits $k_1, \ldots, k_l$
- Encryption function:

$$E_k(m) = m_1 \oplus k_1, \ldots, m_l \oplus k_l = c_1, \ldots, c_l$$

- Decryption function:

$$D_k(c) = c_1 \oplus k_1, \ldots, c_l \oplus k_l = m_1, \ldots, m_l$$

9. Symmetric Encryption

9.5 Stream Ciphers

- The main question in the design of an additive stream cipher is how to expand $k \in \mathcal{K}$ into a potentially infinite key stream $(k_i)_{i \geq 1}$
- Many designs are based on **linear feedback shift registers (LFSRs)**
- In addition to these LFSR-based stream ciphers, there are other stream ciphers (i.e., not based on LFSRs)
- Examples include RC4, Salsa20, and ChaCha (see below)

9. Symmetric Encryption

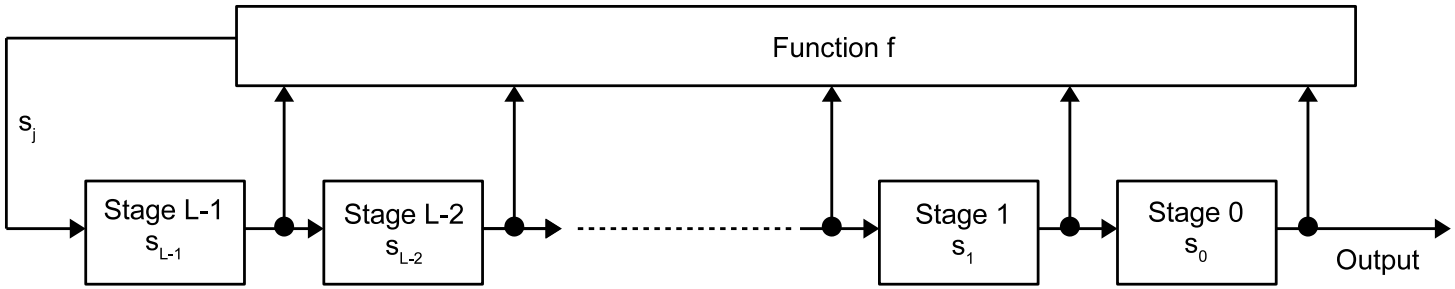
9.5 Stream Ciphers — LFSR-based stream ciphers

- A **feedback shift register (FSR)** of length L comprises:
  - L storage elements (stages), each capable of storing one bit and having one input and one output
  - A clock that controls the movement of data (between the stages) in the FSR
- The stages are initialized with bits $s_0, s_1, \ldots, s_{L-1}$
- This refers to the initial state of the FSR

9. Symmetric Encryption

9.5 Stream Ciphers — LFSR-based stream ciphers

- In each clock cycle, the following operations are performed:
  - The content of stage 0 (i.e., s_0) provides the output
  - The content of stage i (i.e., s_i) is moved to stage $i-1$ for $1 \leq i \leq L-1$
  - The new content of stage $L-1$ is the feedback bit s_j that is computed as $s_j = f(s_0, s_1, \ldots, s_{L-1})$ for some function f



9. Symmetric Encryption

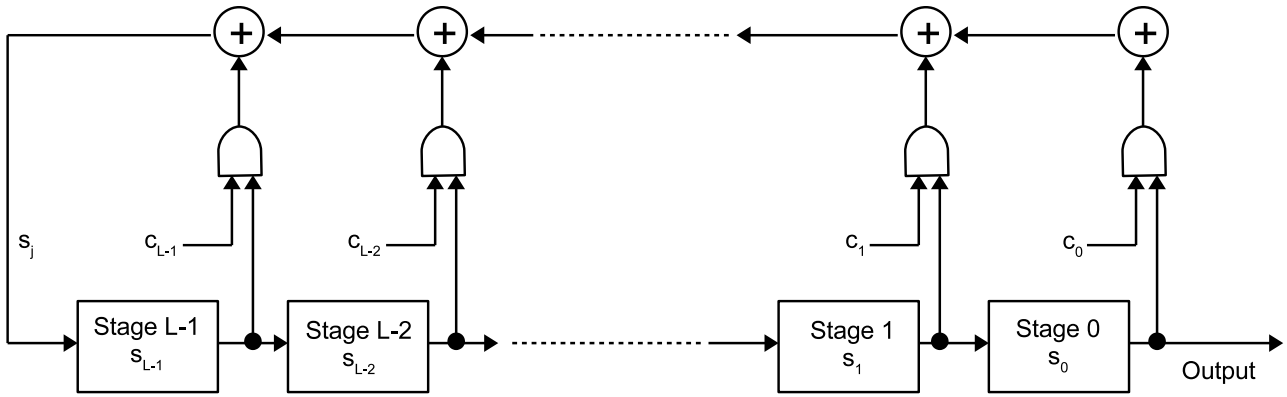
9.5 Stream Ciphers — LFSR-based stream ciphers

- Such an FSR may generate a sequence of (pseudorandom) output values
- Because the length of the FSR (and the number of stages) is finite, the FSR represents a finite state machine (FSM)
- If the register has length L and there are q possible stage values, then the FSR represents an FSM with q^L possible states (and a maximal period of $q^L - 1$)
- If $q = 2$, then there are 2^L possible states (and a maximal period of $2^L - 1$)

9. Symmetric Encryption

9.5 Stream Ciphers — LFSR-based stream ciphers

- If f is the modulo 2 sum of the contents of some stages, the FSR yields a linear FSR (LFSR)
- If the closed semicircles represent AND gates and each c_i $(0 \leq i \leq L-1)$ one bit, then s_j is the modulo 2 sum of the contents of the stages $0 \leq i \leq L-1$ for which $c_i = 1$



9. Symmetric Encryption

9.5 Stream Ciphers — LFSR-based stream ciphers

- An LFSR of length L can be specified as $\langle L, c(X) \rangle$, where

$$c(X) = c_0 + c_1 X + c_2 X^2 + \ldots + c_{L-1} X^{L-1} + X^L$$

  stands for the **connection polynomial** (that is an element of $\mathbb{F}_2[X]$)
- The degree of $c(X)$ is L, meaning that $c_L = 1$.

9. Symmetric Encryption

9.5 Stream Ciphers — LFSR-based stream ciphers

- If $s^{(t=0)} = (s_0, s_1, \ldots, s_{L-1})$ is the initial state of a LFSR with connection polynomial $c(X)$, then the output sequence $s_L, s_{L+1}, s_{L+2}, \ldots$ can be generated as

$$s_{L+t} = \sum_{i=0}^{L-1} c_i s_{i+t} = c_0 s_{i+t} + c_1 s_{i+t+1} + \ldots + c_{L-1} s_{i+t+L-1}$$

  for $t \geq 0$

- The sequence may be infinite, but is cyclic after at most $2^L - 1$ steps

9. Symmetric Encryption

9.5 Stream Ciphers — LFSR-based stream ciphers

- An LFSR of length L has an output sequence with maximum possible period $2^L - 1$, iff its connection polynomial $c(X)$ has degree L and is primitive (and hence irreducible)
- Because LFSRs are well understood in theory and can be implemented very efficiently in hardware, several stream ciphers employ a single LFSR
- Such ciphers are susceptible to known-plaintext attacks (only $2L$ plaintext-ciphertext pairs are required)
- A stream cipher should therefore not directly use the output of a single LFSR

9. Symmetric Encryption

9.5 Stream Ciphers — LFSR-based stream ciphers

- People have proposed several constructions to more securely use LFSRs in the design of a stream cipher
- Examples include the shrinking (or self-shrinking) generator, or — more generally — the use of multiple LFSRs with irregular clocking, such as A5/1 (3 LFSRs), CSS (2 LFSRs), and E0 (4 LFSRs)
- All LFSR-based stream ciphers of the second type have been broken or have serious vulnerabilities and should no longer be used

9. Symmetric Encryption

9.5 Stream Ciphers

- LFSR-based stream ciphers can be implemented efficiently in hardware, but they are not well suited to be implemented in software
- Consequently, there is room for other — preferably additive — stream ciphers optimized for software implementations
- In addition to RC4, Salsa20, and ChaCha, some additional stream ciphers resulted from the eSTREAM project (2004 – 2008)
- Examples include HC-128, Rabbit, SOSEMANUK, Grain, MICKEY, and Trivium (not addressed here)

9. Symmetric Encryption

9.5 Stream Ciphers — RC4

- RC4 is an additive stream cipher originally developed and proposed by Ron Rivest in 1987
- The design was originally kept as a trade secret of RSA Security
- In 1994, the source code of an RC4 implementation was anonymously posted to the Cypherpunks mailing list
- This algorithm is called ARC4 or ARCFOUR
- The correctness of ARC4/ARCFOUR was later confirmed by comparing its outputs to those of some licensed RC4 implementations

9. Symmetric Encryption

9.5 Stream Ciphers — RC4

- RC4 takes a variable-length key k that may range from 1 to 256 bytes (denoted $k[0], \ldots, k[255]$), and it employs a 256-bytes array S of state information (S-box)

Algorithm 9.1 The S-box initialization algorithm of RC4.

(k)

$j = 0$
for $i = 0$ to 255 do $S[i] = i$
for $i = 0$ to 255 do
    $j = (j + S[i] + k[i \bmod |k|]) \bmod 256$
    $S[i] \longleftrightarrow S[j]$

(S)

9. Symmetric Encryption

9.5 Stream Ciphers — RC4

- After the S-box initialization, i and j are set to 0
- RC4 then operates as a PRG to encrypt individual bytes with output byte k (by addition modulo 2)

Algorithm 9.2 The RC4 PRG algorithm.

(S)

$i = (i + 1) \bmod 256$
$j = (j + S[i]) \bmod 256$
$S[i] \longleftrightarrow S[j]$
$k = S[(S[i] + S[j]) \bmod 256]$

(S, k)

9. Symmetric Encryption

9.5 Stream Ciphers — RC4

- RC4 had been used for a very long time, until some weaknesses and statistical defects were found
- Most importantly, the key stream generated by RC4 is biased
- For example, the probability that the second byte being generated is equal to zero is $2/256 = 1/128$ (instead of $1/256$), and the probability that a double zero byte is generated is $1/(256)^2 + 1/(256)^3$ (instead of $1/(256)^2$)
- Biases like these have been exploited in many attacks (e.g., RC4 NOMORE)

9. Symmetric Encryption

9.5 Stream Ciphers — RC4

- RC4 is no longer recommended in most application settings (e.g., RFC 7465 prohibits RC4 for use in SSL/TLS)
- There are a few variants of RC4
  - RC4$^+$
  - RC4A
  - VMPC (Variably Modified Permutation Composition)
  - Spritz
  - ...
- But none of these variants is widely deployed in the field

9. Symmetric Encryption

9.5 Stream Ciphers — Salsa20

- Salsa20 was originally proposed by Dan Bernstein in 2005 (as a submission to the eSTREAM project)
- Like RC4, it is an additive stream cipher
- Unlike RC4, it uses nonces (it is therefore less important to periodically refresh the key)
- There are no published attacks against Salsa20/20 and Salsa20/12
- The best-known attack targets Salsa20/8 (more theoretical than practical)

9. Symmetric Encryption

9.5 Stream Ciphers — Salsa20

- Salsa20 operates on 64-byte (or 512-bit) blocks of data, meaning that a plaintext or ciphertext unit is 64 bytes long
- Encryption function:

$$c = E_k(m, n) = Salsa20_k^{encrypt}(m, n) = m \oplus Salsa20_k^{expand}(n)$$

- $Salsa20^{expand}$ and $Salsa20^{encrypt}$ are keyed with k (typically 32 bytes long) and employ a 16-byte nonce n (of which 8 bytes refer to a counter)
- Furthermore, there is a Salsa20 hash function that takes a 64-byte input value x and hashes it to a 64-byte output value $Salsa20(x)$

9. Symmetric Encryption

9.5 Stream Ciphers — Salsa20

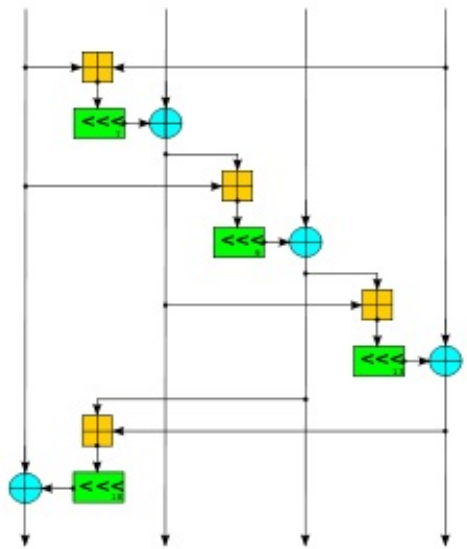
- The Salsa20 hash function is word-oriented, meaning that it operates on words (i.e., 4 bytes or 32 bits)
- Basic operations on words
  - The addition modulo 2^{32} of words w_1 and w_2, denoted as $w_1 + w_2$
  - The addition modulo 2 (XOR) of words w_1 and w_2, denoted as $w_1 \oplus w_2$
  - The c-bit left rotation of word w, denoted as $w \lll c$ for some integer $c \geq 0$

9. Symmetric Encryption

9.5 Stream Ciphers — Salsa20

- The Salsa20 hash function employs several auxiliary functions
- The quarterround function operates on 4-word values $y = (y_0, y_1, y_2, y_3)$ and is defined as $quarterround(y) = z = (z_0, z_1, z_2, z_3)$, where

$$
\begin{aligned}
z_1 &= y_1 \oplus ((y_0 + y_3) \lll 7) \\
z_2 &= y_2 \oplus ((z_1 + y_0) \lll 9) \\
z_3 &= y_3 \oplus ((z_2 + z_1) \lll 13) \\
z_0 &= y_0 \oplus ((z_3 + z_2) \lll 18)
\end{aligned}
$$



9. Symmetric Encryption

9.5 Stream Ciphers — Salsa20

- If y is a 16-word input value $(y_0, y_1, y_2, \ldots, y_{15})$ that can be written as a square matrix

$$\begin{pmatrix} y_0 & y_1 & y_2 & y_3 \\ y_4 & y_5 & y_6 & y_7 \\ y_8 & y_9 & y_{10} & y_{11} \\ y_{12} & y_{13} & y_{14} & y_{15} \end{pmatrix}$$

then the rowround function generates a 16-word output value $z = \mathit{rowround}(y) = (z_0, z_1, z_2, \ldots, z_{15})$, where

$$\begin{aligned} (z_0, z_1, z_2, z_3) &= \mathit{quarterround}(y_0, y_1, y_2, y_3) \\ (z_5, z_6, z_7, z_4) &= \mathit{quarterround}(y_5, y_6, y_7, y_4) \\ (z_{10}, z_{11}, z_8, z_9) &= \mathit{quarterround}(y_{10}, y_{11}, y_8, y_9) \\ (z_{15}, z_{12}, z_{13}, z_{14}) &= \mathit{quarterround}(y_{15}, y_{12}, y_{13}, y_{14}) \end{aligned}$$

9. Symmetric Encryption

9.5 Stream Ciphers — Salsa20

- Similarly, the columnround function takes a 16-word input value $y = (y_0, y_1, y_2, \ldots, y_{15})$ and generates a 16-word output value $z = \mathit{columnround}(y) = (z_0, z_1, z_2, \ldots, z_{15})$, where

$$\begin{aligned}
(z_0, z_4, z_8, z_{12}) &= \mathit{quarterround}(y_0, y_4, y_8, y_{12}) \\
(z_5, z_9, z_{13}, z_1) &= \mathit{quarterround}(y_5, y_9, y_{13}, y_1) \\
(z_{10}, z_{14}, z_2, z_6) &= \mathit{quarterround}(y_{10}, y_{14}, y_2, y_6) \\
(z_{15}, z_3, z_7, z_{11}) &= \mathit{quarterround}(y_{15}, y_3, y_7, y_{11})
\end{aligned}$$

9. Symmetric Encryption

9.5 Stream Ciphers — Salsa20

- The rowround and columnround functions are combined in a doubleround function
- If $y = (y_0, y_1, y_2, \ldots, y_{15})$ is a 16-word input value, then

$$\begin{aligned} z &= (z_0, z_1, z_2, \ldots, z_{15}) \\ &= \mathit{doubleround}(y) \\ &= \mathit{rowround}(\mathit{columnround}(y)) \end{aligned}$$

is the respective 16-word output value

9. Symmetric Encryption

9.5 Stream Ciphers — Salsa20

- The littleendian function encodes a word or 4-byte sequence $b = (b_0, b_1, b_2, b_3)$ in little-endian order (b_3, b_2, b_1, b_0) that represents the value $b_3 \cdot 2^{24} + b_2 \cdot 2^{16} + b_1 \cdot 2^8 + b_0$ (typically written in hexadecimal notation)
- For example, $\mathit{littleendian}(86, 75, 30, 9) = (9, 30, 75, 86)$ represents $9 \cdot 2^{24} + 30 \cdot 2^{16} + 75 \cdot 2^8 + 86$ and can be written as 0x091E4B56
- The littlendian function has an inverse function, so $\mathit{littleendian}^{-1}(0\text{x}091\text{E}4\text{B}56) = (86, 75, 30, 9)$

9. Symmetric Encryption

9.5 Stream Ciphers — Salsa20

- The Salsa20 hash function takes a 64-byte input value $x = (x[0], x[1], \ldots, x[63])$ that refers to 16 words

$$\begin{aligned} x_0 &= \mathit{littlendian}(x[0], x[1], x[2], x[3]) \\ x_1 &= \mathit{littlendian}(x[4], x[5], x[6], x[7]) \\ &\ldots \\ x_{15} &= \mathit{littlendian}(x[60], x[61], x[62], x[63]) \end{aligned}$$

- It then computes

$$z = (z_0, z_1, z_2, \ldots, z_{15}) = \mathit{doubleround}^{10}(x_0, x_1, \ldots, x_{15})$$

9. Symmetric Encryption

9.5 Stream Ciphers — Salsa20

- Finally, the output of $Salsa20(x)$ is the concatenation of the 16 words that are generated as follows:

$$littlendian^{-1}(z_0 + x_0)$$
$$littlendian^{-1}(z_1 + x_1)$$
$$\ldots$$
$$littlendian^{-1}(z_{15} + x_{15})$$

- The 20 rounds of Salsa20 come from the fact that the doubleround function is iterated 10 times, and each iteration represents two rounds (one for the columnround function and one for the rowround function)

9. Symmetric Encryption

9.5 Stream Ciphers — Salsa20

- As its name suggests, the aim of the Salsa20 expansion function is to expand a 16-byte input n into a 64-byte output
- It therefore uses 32 or 16 bytes of keying material and 16 constant bytes (4 constant words)
- Depending on whether the keying material consists of 32 or 16 bytes, the constant words and the respective expansion functions are different

9. Symmetric Encryption

9.5 Stream Ciphers — Salsa20

- Case 1: If the keying material consists of 32 bytes, then this material is split into two halves that represent two 16-bytes keys k_0 and k_1
- In this case, the constant words are as follows

$$\begin{aligned} \sigma_0 &= (101, 120, 112, 97) = 0\text{x}61707865 \\ \sigma_1 &= (110, 100, 32, 51) = 0\text{x}3320646\text{E} \\ \sigma_2 &= (50, 45, 98, 121) = 0\text{x}79622\text{D}32 \\ \sigma_3 &= (116, 101, 32, 107) = 0\text{x}6\text{B}206574 \end{aligned}$$

9. Symmetric Encryption

9.5 Stream Ciphers — Salsa20

- The Salsa20 expansion function is then defined as

$$Salsa20_{k_0,k_1}(n) = Salsa20(\sigma_0, k_0, \sigma_1, n, \sigma_2, k_1, \sigma_3)$$

Note that $littleendian(\sigma_0) = littleendian(101, 120, 112, 97) = 0x61707865$, so the argument that is subject to the Salsa20 hash function always starts with the four bytes 0x61, 0x70, 0x78, and 0x65

9. Symmetric Encryption

9.5 Stream Ciphers — Salsa20

- Case 2: If the keying material consists of 16 bytes, then this material represents a single 16-byte key k that is applied twice
- In this case, a slightly different set of 4-byte τ constants is used (the two different bytes are underlined)

$$\begin{aligned} \tau_0 &= (101, 120, 112, 97) \\ \tau_1 &= (110, 100, 32, \underline{49}) \\ \tau_2 &= (\underline{54}, 45, 98, 121) \\ \tau_3 &= (116, 101, 32, 107) \end{aligned}$$

- The Salsa20 expansion function is then defined as

$$Salsa20_k(n) = Salsa20(\tau_0, k, \tau_1, n, \tau_2, k, \tau_3)$$

9. Symmetric Encryption

9.5 Stream Ciphers — Salsa20

- σ and τ can be seen as constants c, k is the key, and n is the argument of the Salsa20 expansion function
- The input can be written in a specific matrix layout

$$\begin{pmatrix} c & k & k & k \\ k & c & n & n \\ n & n & c & k \\ k & k & k & c \end{pmatrix}$$

- This layout is somewhat arbitrary and can be changed at will (e.g., ChaCha uses a different layout)

9. Symmetric Encryption

9.5 Stream Ciphers — Salsa20

- Salsa20 is an additive stream cipher, meaning that an appropriately sized key stream is generated and added modulo 2 to the plaintext message
- The Salsa20 expansion function is used to generate the key stream
- Let k be a 32- or 16-byte key, n an 8-byte nonce, and m an l-byte plaintext message (where $0 \leq l \leq 2^{70}$)
- The Salsa20 encryption of m with nonce n under key k, denoted as $Salsa20_k(m, n)$, is computed as $m \oplus Salsa20_k(n')$, where $Salsa20_k(n')$ represents the key stream and n' is derived from n by adding an 8-byte counter

9. Symmetric Encryption

9.5 Stream Ciphers — Salsa20

- More specifically, the key stream is constructed as

$$Salsa20_k(n \parallel 0) \parallel Salsa20_k(n \parallel 1) \parallel \ldots \parallel Salsa20_k(n \parallel 2^{64} - 1)$$

- The Salsa20 encryption function can be expressed as

$$\begin{aligned} c &= (c[0], c[1], \ldots, c[l-1]) \\ &= (m[0], m[1], \ldots, m[l-1]) \oplus Salsa20_k(n') \end{aligned}$$

9. Symmetric Encryption

9.5 Stream Ciphers — Salsa20

- Since Salsa20 is an additive stream cipher, the encryption and decryption functions are the same
- Because the length of a nonce is controversially discussed in the community, Bernstein proposed a variant of Salsa20 that can handle longer nonces
- More specifically, XSalsa20 can take nonces that are 192 bits long (instead of only 64 bits)
- The XSalsa20 encryption and decryption algorithms are slightly different, but the security level is provably the same

9. Symmetric Encryption

9.5 Stream Ciphers — ChaCha

- In 2008, Bernstein proposed a modified version of the Salsa20 stream cipher named ChaCha
- Again, the term refers to a family of stream ciphers that comprises ChaCha20 (20 rounds), ChaCha12 (12 rounds), and ChaCha8 (8 rounds)
- ChaCha is structurally identical to Salsa20, but it uses a different round function and a different matrix layout
- Furthermore, it uses a key that is always 32 bytes (256 bits) long, a nonce that is 12 bytes (96 bits) long, and a block counter that is only 4 bytes (32 bits) long

9. Symmetric Encryption

9.5 Stream Ciphers — ChaCha

- Instead of using $y = (y_0, y_1, y_2, y_3)$ and $z = (z_0, z_1, z_2, z_3)$, ChaCha uses four 32-bit words a, b, c, and d
- This means that

$$\begin{aligned} z_1 &= y_1 \oplus ((y_0 + y_3) \overleftarrow{\circlearrowright} 7) \\ z_2 &= y_2 \oplus ((z_1 + y_0) \overleftarrow{\circlearrowright} 9) \\ z_3 &= y_3 \oplus ((z_2 + z_1) \overleftarrow{\circlearrowright} 13) \\ z_0 &= y_0 \oplus ((z_3 + z_2) \overleftarrow{\circlearrowright} 18) \end{aligned}$$

is written as

$$\begin{aligned} b &= b \oplus ((a + d) \overleftarrow{\circlearrowright} 7) \\ c &= c \oplus ((b + a) \overleftarrow{\circlearrowright} 9) \\ d &= d \oplus ((c + b) \overleftarrow{\circlearrowright} 13) \\ a &= a \oplus ((d + c) \overleftarrow{\circlearrowright} 18) \end{aligned}$$

9. Symmetric Encryption

9.5 Stream Ciphers — ChaCha

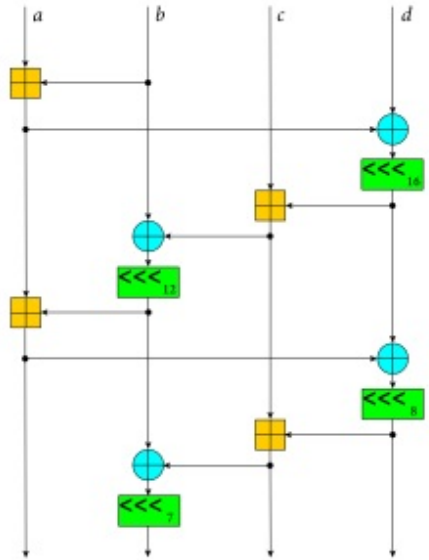
- The operations performed by ChaCha are the same as the ones performed by Salsa20, but they are applied in a different order and each word is updated twice (instead of just once)
- The advantage is that the ChaCha round function provides more diffusion than the Salsa20 round function
- Also, the rotation distances are changed from 7, 9, 13, and 18 to 16, 12, 8, and 7 (but this difference is less important)

9. Symmetric Encryption

9.5 Stream Ciphers — ChaCha

- The ChaCha quarterround function updates a, b, c, and d as follows:

$$
\begin{aligned}
a &= a + b; \quad d = d \oplus a; \quad d = d \hookleftarrow 16;\\
c &= c + d; \quad b = b \oplus c; \quad b = b \hookleftarrow 12;\\
a &= a + b; \quad d = d \oplus a; \quad d = d \hookleftarrow 8;\\
c &= c + d; \quad b = b \oplus c; \quad b = b \hookleftarrow 7;
\end{aligned}
$$



9. Symmetric Encryption

9.5 Stream Ciphers — ChaCha

- Using the quarterround function, ChaCha combines columnround and "diagonalround" (no rowround) in every round

Algorithm 9.3 The ChaCha20 PRG

(k, i, n)

$S = \sigma \parallel k \parallel i \parallel n$
$S' = S$
for $i = 1$ to 10 do begin
  $quarterround(S'_0, S'_4, S'_8, S'_{12})$
  $quarterround(S'_1, S'_5, S'_9, S'_{13})$
  $quarterround(S'_2, S'_6, S'_{10}, S'_{14})$
  $quarterround(S'_3, S'_7, S'_{11}, S'_{15})$
  $quarterround(S'_0, S'_5, S'_{10}, S'_{15})$
  $quarterround(S'_1, S'_6, S'_{11}, S'_{12})$
  $quarterround(S'_2, S'_7, S'_8, S'_{13})$
  $quarterround(S'_3, S'_4, S'_9, S'_{14})$
end
$S = S + S'$

Serialized(S)

9. Symmetric Encryption

9.5 Stream Ciphers — ChaCha

- Initialization parameters
  - Four 4-byte constants σ_0, σ_1, σ_2, and σ_3 (the same as Salsa20)
  - 32-byte key k
  - 4-byte counter i
  - 12-byte nonce n

Matrix layout of S

$$\begin{pmatrix} c & c & c & c \\ k & k & k & k \\ k & k & k & k \\ n & n & n & n \end{pmatrix}$$

9. Symmetric Encryption

9.5 Stream Ciphers — ChaCha

- Similar to Salsa20, no practically relevant cryptanalytical attack against ChaCha (ChaCha20) is known to exist
- ChaCha20 is widely used on the Internet (mainly as a replacement for RC4)
- Most importantly, it is often combined with Bernstein's Poly1305 message authentication code to provide authenticated encryption

9. Symmetric Encryption

9.6 Block Ciphers

- Every practical symmetric encryption system processes plaintext messages unit by unit
- In the case of a block cipher such a unit is called "block"
- Consequently, a block cipher maps plaintext message blocks of a specific length into ciphertext blocks of typically the same length
- $\mathcal{M} = \mathcal{C} = \Sigma^n$ for some alphabet Σ and block length n
- In a typical setting, n is 128 or 256 bits

9. Symmetric Encryption

9.6 Block Ciphers

- A permutation on set S is a bijective function $f : S \to S$
- If one fixes a block length n and works with $\mathcal{M} = \mathcal{C} = \Sigma^n$, then any element π randomly selected from $\mathrm{Perms}[\Sigma^n]$ defines a block cipher
- The encryption and decryption functions (E_π and D_π) can be defined as follows:

$$E_\pi : \begin{array}{l} \Sigma^n \longrightarrow \Sigma^n \\ w \longmapsto \pi(w) \end{array} \qquad D_\pi : \begin{array}{l} \Sigma^n \longrightarrow \Sigma^n \\ w \longmapsto \pi^{-1}(w) \end{array}$$

- There are $|P(\Sigma^n)| = (\Sigma^n)!$ elements in $\mathrm{Perms}[\Sigma^n]$

9. Symmetric Encryption

9.6 Block Ciphers

- There are $(2^n)!$ possible permutations in $\mathrm{Perms}[\{0,1\}^n]$ (i.e., $\Sigma = \{0,1\}$)
- For a typical block length n of 64 bits, there are

$$2^{64}! = 18,446,744,073,709,551,616!$$

  possible permutation
- This number requires $> 2^{69}$ bits to encode it
- Block ciphers are usually designed to take a reasonably long key and still generate a permutation that looks random

9. Symmetric Encryption

9.6 Block Ciphers

- This means that one uses only some possible permutations of Σ^n (from $\mathrm{Perms}[\Sigma^n]$) as encryption and decryption functions and comparably short keys to refer to them
- Hence, a block cipher is a PRP that uses a key k from a moderately sized key space $\mathcal{K}$ to define a family of bijective encryption functions $E_k : \Sigma^n \to \Sigma^n$ and a family of respective decryption functions $D_k : \Sigma^n \to \Sigma^n$
- To analyze the security of such a block cipher, one has to study the algebraic properties of the PRP

9. Symmetric Encryption

9.6 Block Ciphers

- The design of a block cipher combines permutations and substitutions to generate confusion and diffusion
  - **Confusion** is to make the relation between the key and the ciphertext as complex as possible
  - **Diffusion** is to spread the influence of a single plaintext bit over many ciphertext bits
- Block ciphers that combine permutations and substitutions in multiple rounds are called **substitution-permutation ciphers**
- The **Data Encryption Standard (DES)** is the prime example of such a substitution-permutation cipher

9. Symmetric Encryption

9.6 Block Ciphers — DES

- In the early 1970s, the U.S. National Bureau of Standards (NBS) set up a competition for a standardized block cipher
- IBM submitted a block cipher called **Lucifer**
- NBS, NSA, and IBM refined the design of Lucifer and finally standardized the result as DES in FIPS PUB 46 (1977).
- The standard was reaffirmed by NIST in 1983, 1988, 1993, and 1999, before it was officially withdrawn in July 2004
- The Triple Data Encryption Algorithm (TDEA) or 3DES is still used in situations that can handle moderate performance

9. Symmetric Encryption

9.6 Block Ciphers — DES

- DES is a substitution-permutation cipher, but it is also a **Feistel cipher**
- A Feistel cipher is a block cipher with a characteristic structure (aka Feistel network)
- The alphabet is $\Sigma = \mathbb{Z}_2 = \{0, 1\}$ and the block length is $2t$ for a reasonably sized $t \in \mathbb{N}^+$
- The Feistel cipher runs in $0 < r \in \mathbb{N}$ rounds, where r round keys $k_1, \ldots, k_r$ are derived from $k \in \mathcal{K}$ and used on a per-round basis

9. Symmetric Encryption

9.6 Block Ciphers — DES

- The encryption function E_k starts by splitting the plaintext message block m into two halves of t bits each
- Let L_0 be the left half and R_0 the right half of m, i.e., $m = L_0 \parallel R_0 = (L_0, R_0)$
- For $i = 1, \ldots, r$, a sequence of pairs (L_i, R_i) is recursively computed as

$$(L_i, R_i) = (R_{i-1}, L_{i-1} \oplus f_{k_i}(R_{i-1}))$$

- This means that $L_i = R_{i-1}$ and $R_i = L_{i-1} \oplus f_{k_i}(R_{i-1})$

9. Symmetric Encryption

9.6 Block Ciphers — DES

- The pair (L_r, R_r) in reverse order then represents the ciphertext block c

$$E_k(m) = E_k(L_0, R_0) = (R_r, L_r) = c$$

- The recursive formula from above can be written as

$$(L_{i-1}, R_{i-1}) = (R_i \oplus f_{k_i}(L_i), L_i)$$

- It is thus possible to recursively compute L_{i-1} and R_{i-1} from L_i, R_i, and k_i, and to determine (L_0, R_0) from (R_r, L_r) using the round keys in reverse order (i.e., $k_r, \ldots, k_1$)

9. Symmetric Encryption

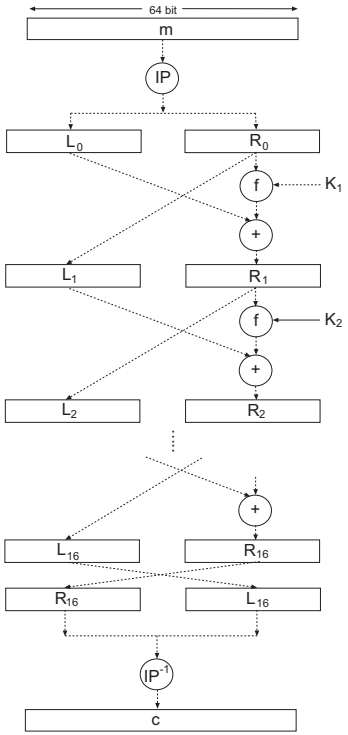
9.6 Block Ciphers — DES

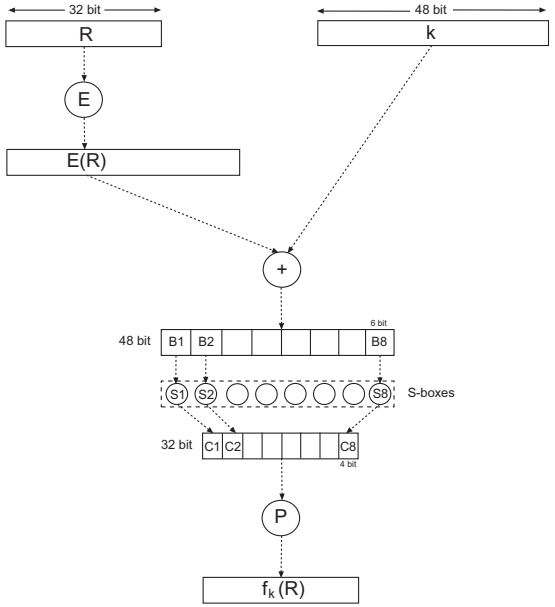
- DES is a Feistel network with $t = 32$ and $r = 16$
- This means that the block length is 64 bits, and hence $\mathcal{M} = \mathcal{C} = \{0,1\}^{64}$, and that the DES encryption and decryption algorithms operate in 16 rounds
- Furthermore, DES keys are 64-bit strings, where the last bit of each byte is set to have odd parity

$$\mathcal{K} = \{(k_1, \ldots, k_{64}) \in \{0,1\}^{64} \mid \sum_{i=1}^{8} k_{8j+i} \equiv 1 \pmod 2 \text{ for } j = 0, \ldots, 7\}$$

9. Symmetric Encryption

9.6 Block Ciphers — DES





Algorithm 9.4 The DES encryption algorithm

(m, k)

$m = IP(m)$
$L_0 = m|_{32}$
$R_0 = m|^{32}$
for $i = 1$ to 16 do
$\quad L_i = R_{i-1}$
$\quad R_i = L_{i-1} \oplus f_{k_i}(R_{i-1})$
$c = IP^{-1}(R_{16}, L_{16})$

(c)

9. Symmetric Encryption

9.6 Block Ciphers — DES

- Since its standardization, the DES has been subject to a lot of public scrutiny
- People have found 4 weak keys and 12 semiweak keys
  - A DES key k is **weak** if $DES_k(DES_k(m)) = m$ for all $m \in \mathcal{M} = \{0,1\}^{64}$
  - The DES keys k_1 and k_2 are **semiweak** if $DES_{k_1}(DES_{k_2}(m)) = m$ for all $m \in \mathcal{M} = \{0,1\}^{64}$
- Weak and semiweak DES keys should not be used in practice
- Since there are only $16 = 2^4$ such keys, the probability of randomly generating one is very small

$$\frac{2^4}{2^{56}} = 2^{-52} \approx 2.22 \cdot 10^{-16}$$

9. Symmetric Encryption

9.6 Block Ciphers — DES

- Several cryptanalytical attacks have been developed in an attempt to break DES
- The following attacks were published in the early 1990s
  - **Differential cryptanalysis** represents a CPA that requires 2^{47} chosen plaintexts
  - **Linear cryptanalysis** represents a known-plaintext attack that requires 2^{43} known plaintexts
- All newly proposed block ciphers are routinely shown to be resistant against differential and linear cryptanalysis

9. Symmetric Encryption

9.6 Block Ciphers — DES

- From a practical viewpoint, the major vulnerability and security problem of DES is its relatively small key length (and key space)
- Note that a DES key is effectively 56 bits long, and hence the key space comprises $2^{56} = 72,057,594,037,927,936$ elements
- Consequently, a key search is successful after 2^{56} trials in the worst case and $2^{56}/2 = 2^{55}$ trials on the average
- Due to the complementation property, 2^{54} trials are sufficient on average

9. Symmetric Encryption

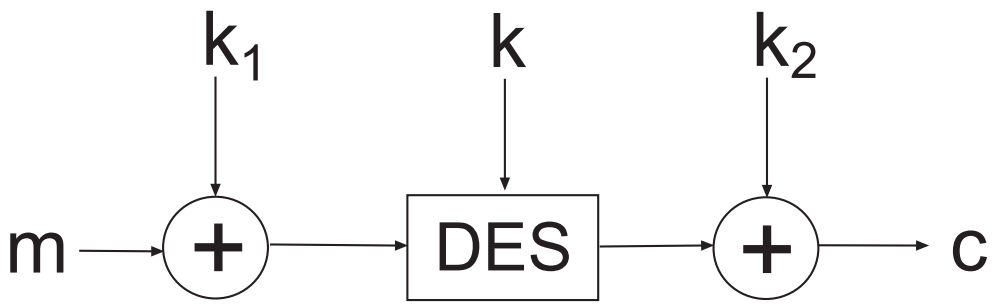
9.6 Block Ciphers — DES

- The feasibility of an exhaustive key search was first publicly discussed by Diffie and Hellman in 1977
- They estimated that a brute-force machine that could find a DES key within a day would cost 20 million USD (with a lot of room for time-memory trade-offs)
- Michael J. Wiener later proposed the design of several dedicated machines to find DES keys
- In 1998, the Electronic Frontier Foundation (EFF) built a DES search machine named **Deep Crack**
- More recently, a massively parallel machine called **COPACOBANA** was built for 10,000 USD

9. Symmetric Encryption

9.6 Block Ciphers — DES

- **DESX** is a modified version of DES that compensates for its relatively small key length
- It employs a technique called **key whitening**
- DESX is practically relevant and employed by the Encrypted File System (EFS) in Microsoft Windows



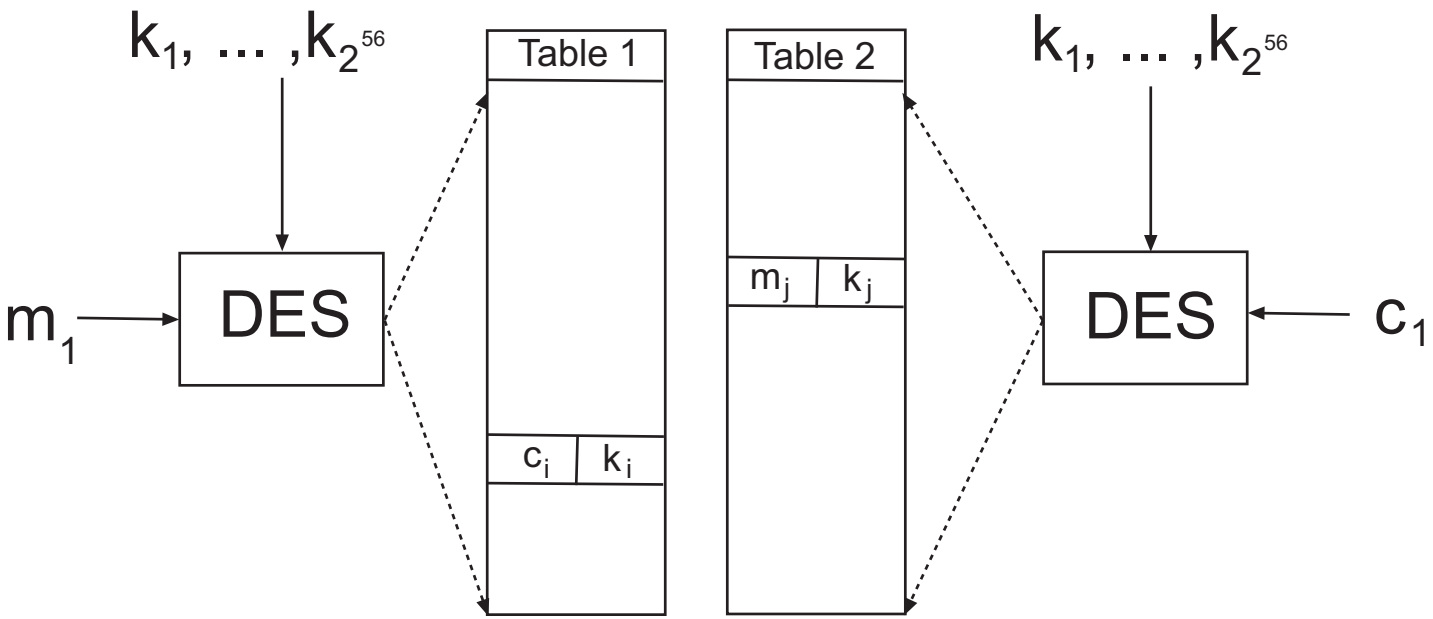
9. Symmetric Encryption

9.6 Block Ciphers — DES

- Another possibility to address (or solve) the small key length problem is to iterate DES multiple times
- Important remarks
  - Multiple iterations with the same key are pointless (i.e., different keys are needed)
  - The DES encryption functions are not closed with regard to concatenation (i.e., they do not form a group)
- The first (meaningful) possibility to iterate the DES is double encryption with independent keys
- This is susceptible to a **meet-in-the-middle attack**

9. Symmetric Encryption

9.6 Block Ciphers — DES



9. Symmetric Encryption

9.6 Block Ciphers — DES

- Due to the meet-in-the-middle attack, one usually iterates DES three times (Triple DES or 3DES)
- FIPS PUB 46-3 specifies the TDEA, and this specification also conforms to ANSI X9.52
- A TDEA key consists of three keys that are collectively referred to as a key bundle, i.e., $k = (k_1, k_2, k_3)$
- TDEA encryption function:

$$c = E_{k_3}(D_{k_2}(E_{k_1}(m)))$$

- Consequently, a TDEA or 3DES encryption is also referred to as EDE (encrypt-decrypt-encrypt)

9. Symmetric Encryption

9.6 Block Ciphers — AES

- Between 1997 and 2000, NIST carried out a competition to standardize a successor for the DES, called the AES
- Many parties from industry and academia participated in the competition
- 15 submissions qualified as AES candidates
- NIST selected 5 finalists
  - MARS
  - RC6
  - Rijndael
  - Serpent
  - Twofish

9. Symmetric Encryption

9.6 Block Ciphers — AES

- In October 2000, NIST decided that **Rijndael** would become the AES (FIPS PUB 197)
- It was selected mainly because of its ease of implementation in hardware and its strong performance on nearly all platforms
- According to the requirements, the AES is a block cipher with a block length of 128 bits and a variable key length of 128 (AES-128), 192 (AES-192), or 256 bits (AES-256)
- The number of rounds depends on the key length (i.e., 10, 12, or 14 rounds)

9. Symmetric Encryption

9.6 Block Ciphers — AES

- The AES is byte oriented
- Each byte represents an element of $\mathbb{F}_{2^8}$ (or $GF(2^8)$) and can be written in binary or hexadecimal notation
- Alternatively, the 8 bits of a byte can also be seen and written as the coefficients of a polynomial of degree 7:

$$b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x + b_0 = \sum_{i=0}^{7} b_ix^i$$

- For example, the byte $\{10100011\} = 0\text{xA3}$ can be written as polynomial $x^7 + x^5 + x + 1$

9. Symmetric Encryption

9.6 Block Ciphers — AES

- The AES operates in the binary extension field $\mathbb{F}_{2^8}$
- This means that the elements of the field are polynomials over $\mathbb{F}_2$ ($\mathbb{Z}_2$) with degree equal or smaller than 7
- Using these polynomials, one can add and multiply field elements
- The multiplication operation must be specified in the polynomial notation
- It refers to the multiplication of two polynomials over $\mathbb{Z}_2$ modulo an irreducible polynomial of degree 8
- In the case of the AES, this polynomial is

$$f(x) = x^8 + x^4 + x^3 + x + 1$$

9. Symmetric Encryption

9.6 Block Ciphers — AES

- Mathematically speaking, $\mathbb{Z}_2[x]_{f(x)}$ is a field if $f(x)$ is an irreducible polynomial over $\mathbb{Z}_2$
- In the case of AES, $f(x)$ is an irreducible polynomial over $\mathbb{Z}_2$ with degree 8
- It follows that $\mathbb{Z}_2[x]_{f(x)}$ is a field, called *AES field*, that is isomorphic to $\mathbb{F}_{2^8}$
- Because it is a field, every nonzero element $b(x)$ has a multiplicative inverse element $b^{-1}(x)$
- As for any field, this element can be efficiently computed with the extended Euclid algorithm

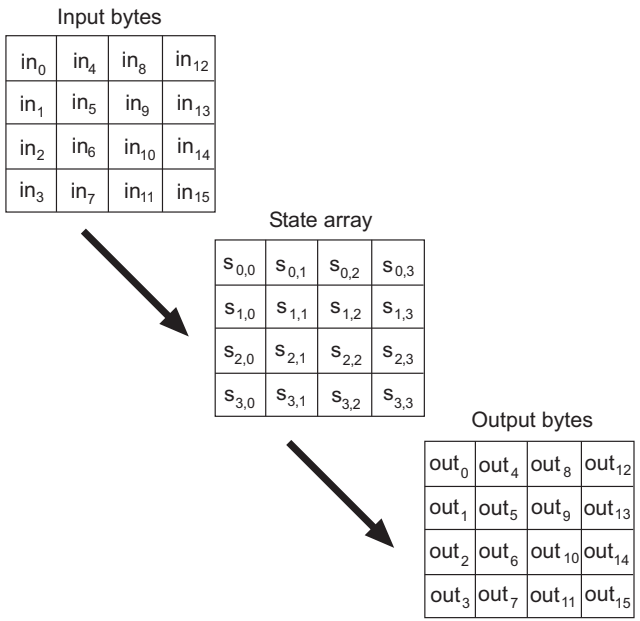
9. Symmetric Encryption

9.6 Block Ciphers — AES

- Internally, the AES operates on a two-dimensional array s of bytes, called the **State**
- The State consists of 4 rows and N_b columns (where $N_b = 4$ for all versions of the AES)
- Each entry in the State refers to a byte $s_{r,c}$ or $s[r, c]$, where $0 \leq r < 4$ refers to the row and $0 \leq c < 4$ refers to the column
- The State can be viewed as a two-dimensional 4x4 array (or matrix) of bytes or as a one-dimensional array of four 32-bit words are equivalent

9. Symmetric Encryption

9.6 Block Ciphers — AES



Algorithm 9.5 The AES encryption algorithm

(in)

$s = in$
$s =$ AddRoundKey$(s, w[0, N_b - 1])$
for $r = 1$ to $(N_r - 1)$ do
  $s =$ SubBytes(s)
  $s =$ ShiftRows(s)
  $s =$ MixColumns(s)
  $s =$ AddRoundKey$(s, w[rN_b, (r+1)N_b - 1])$
$s =$ SubBytes(s)
$s =$ ShiftRows(s)
$s =$ AddRoundKey$(s, w[N_r N_b, (N_r + 1)N_b - 1])$
$out = s$

(out)

9. Symmetric Encryption

9.6 Block Ciphers — AES

- The SubBytes() transformation stands for a substitution cipher in which each byte $s_{r,c}$ of the State is replaced with another byte $s'_{r,c}$ from a substitution table (S-box)
- The substitution cipher defined by the AES' S-box is bijective (one-to-one) and nonlinear (i.e., SubBytes(s) + SubBytes(s') $\neq$ SubBytes(s + s') for two states s and s')
- The S-box is the only nonlinear component of the AES
- As such, it is important from a security viewpoint

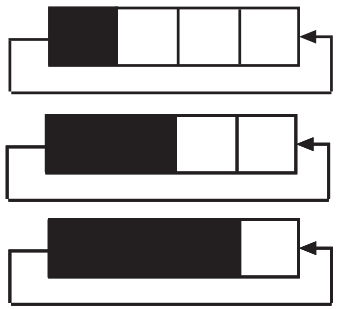
9. Symmetric Encryption

9.6 Block Ciphers — AES

- The ShiftRows() transformation cyclically shifts (or rotates) the bytes in each row of the State

s

$s_{0,0}$	$s_{0,1}$	$s_{0,2}$	$s_{0,3}$
$s_{1,0}$	$s_{1,1}$	$s_{1,2}$	$s_{1,3}$
$s_{2,0}$	$s_{2,1}$	$s_{2,2}$	$s_{2,3}$
$s_{3,0}$	$s_{3,1}$	$s_{3,2}$	$s_{3,3}$



s'

$s_{0,0}$	$s_{0,1}$	$s_{0,2}$	$s_{0,3}$
$s_{1,1}$	$s_{1,2}$	$s_{1,3}$	$s_{1,0}$
$s_{2,2}$	$s_{2,3}$	$s_{2,0}$	$s_{2,1}$
$s_{3,3}$	$s_{3,0}$	$s_{3,1}$	$s_{3,2}$

9. Symmetric Encryption

9.6 Block Ciphers — AES

- The MixColumns() transformation operates on each column of the State individually, and it subjects each column to a linear transformation
- When the MixColumns() transformation operates on column c ($0 \leq c < 4$), it considers the 4 bytes $s_{0,c}, s_{1,c}, s_{2,c}$, and $s_{3,c}$ of the State simultaneously
- The transformation is defined as follows:

$$\begin{pmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{pmatrix} = \begin{pmatrix} 0x02 & 0x03 & 0x01 & 0x01 \\ 0x01 & 0x02 & 0x03 & 0x01 \\ 0x01 & 0x01 & 0x02 & 0x03 \\ 0x03 & 0x01 & 0x01 & 0x02 \end{pmatrix} \cdot \begin{pmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{pmatrix}$$

9. Symmetric Encryption

9.6 Block Ciphers — AES

- In the AddRoundKey() transformation, a word of the key schedule w is added modulo 2 to each column of the State
- This means that

$$[s'_{0,c}, s'_{1,c}, s'_{2,c}, s'_{3,c}] = [s_{0,c}, s_{1,c}, s_{2,c}, s_{3,c}] \oplus w[rN_b + c]$$

  for $0 \leq c < N_b$ and $0 \leq r \leq N_r$
- There is a unique key expansion algorithm (not addressed here)

9. Symmetric Encryption

9.6 Block Ciphers — AES

Algorithm 9.7 The AES decryption algorithm

(in)

$s = in$
$s = \text{AddRoundKey}(s, w[N_r N_b, (N_r + 1)N_b - 1])$
for $r = N_r - 1$ downto 1 do
  $s = \text{AddRoundKey}(s, w[rN_b, (r+1)N_b - 1])$
  $s = \text{InvMixColumns}(s)$
  $s = \text{InvShiftRows}(s)$
  $s = \text{InvSubBytes}(s)$
$s = \text{AddRoundKey}(s, w[0, N_b - 1])$
$s = \text{InvShiftRows}(s)$
$s = \text{InvSubBytes}(s)$
$out = s$

(out)

9. Symmetric Encryption

9.6 Block Ciphers — AES

- In 2003, the NSA approved the AES to become a legitimate cipher to encrypt classified information

  "The design and strength of all key lengths of the AES algorithm (i.e., 128, 192 and 256) are sufficient to protect classified information up to the SECRET level. TOP SECRET information will require use of either the 192 or 256 key lengths."

  [. . .]

  "The implementation of AES in products intended to protect national security systems and/or information must be reviewed and certified by NSA prior to their acquisition and use."

9. Symmetric Encryption

9.6 Block Ciphers — AES

- This announcement marks the first time that the public has access to a cipher approved by NSA for the encryption of information classified as TOP SECRET (at least, if the longer-key versions of the AES are used).
- Unfortunately, one doesn't know what the NSA currently knows about the security of the AES
- Outside the NSA, the AES has been subject to a lot of public scrutiny
- Even after two decades of cryptanalytical research, nobody has found a vulnerability that can be turned into a practical attack against the AES

9. Symmetric Encryption

9.7 Modes of Operation

- If one wants to encrypt long messages, then one needs a block cipher and an appropriate mode of operation
- Historically, the most important document was FIPS PUB 81 published by NIST in 1980
- The document specifies four modes of operation (for DES)
  - Electronic code book (ECB)
  - Cipherblock chaining (CBC)
  - Output feedback (OFB)
  - Cipher feedback (CFB)
- In 2001, NIST added counter (CTR) mode

9. Symmetric Encryption

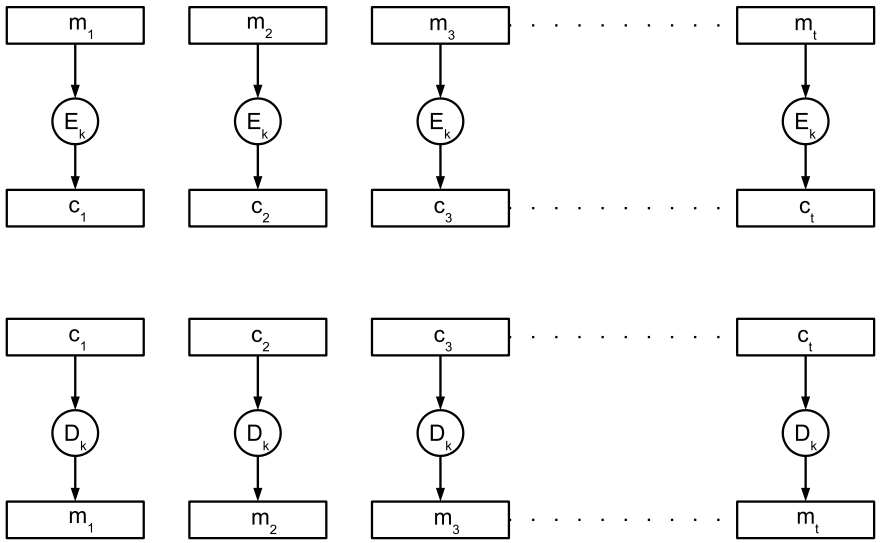
9.7 Modes of Operation

- All 5 modes are confidentiality modes, i.e., they protect (only) the confidentiality of messages
- They neither provide the authenticity nor the integrity of messages
- Meanwhile, cryptography has come up with modes of operation that provide support for AE and AEAD
- They have become very important in the field ($\rightarrow$ Chapter 11)

9. Symmetric Encryption

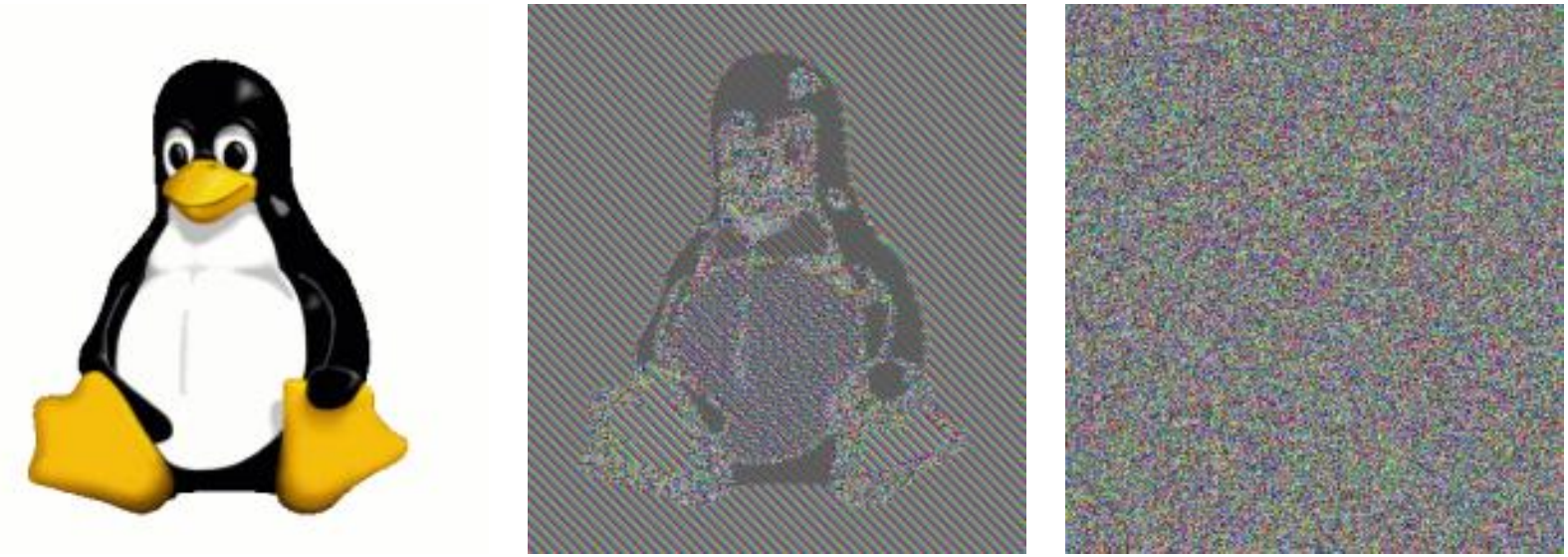
9.7 Modes of Operation — ECB

- The ECB mode is the simplest and most straightforward mode of operation for a block cipher (it should no longer be used)



9. Symmetric Encryption

9.7 Modes of Operation — ECB



© https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation#Electronic_codebook_(ECB)

9. Symmetric Encryption

9.7 Modes of Operation — CBC

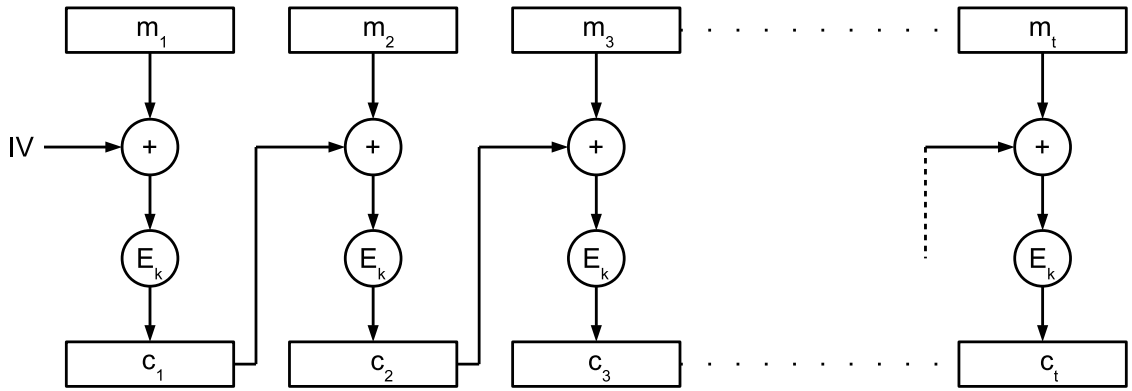
- The CBC mode of operation is often used to overcome the most important disadvantages of ECB
- The encryption of a plaintext message block m_i not only depends on m_i and k, but also on the previous ciphertext block c_{i-1} (or the IV in case of m_1)
- This means that the ciphertext blocks are cryptographically chained
- The use of an IV turns the encryption function into a probabilistic one
- The IV need not be kept secret, but it must be unpredictable (e.g., BEAST attack)

9. Symmetric Encryption

9.7 Modes of Operation — CBC

- Encryption function:

$$\begin{aligned} c_0 &= IV \\ c_i &= E_k(m_i \oplus c_{i-1}) \text{ for } 1 \leq i \leq t \end{aligned}$$

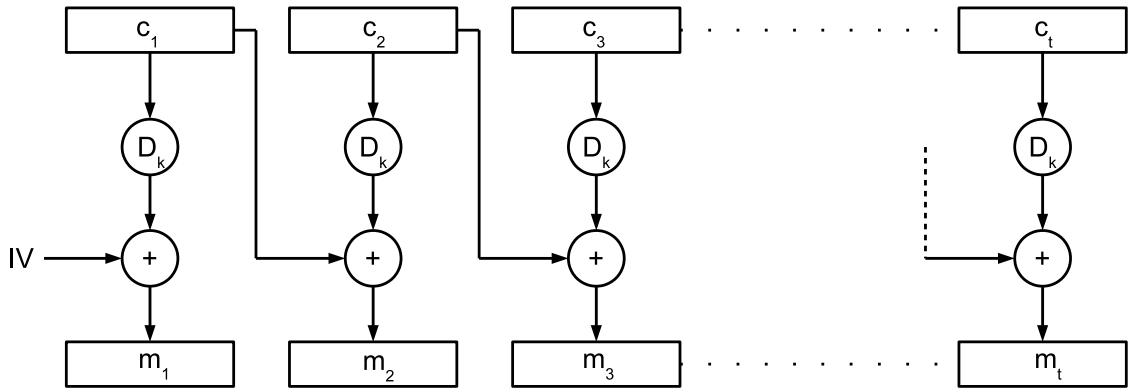


9. Symmetric Encryption

9.7 Modes of Operation — CBC

- Decryption function:

$$
\begin{aligned}
c_0 &= IV \\
m_i &= D_k(c_i) \oplus c_{i-1} \text{ for } 1 \leq i \leq t
\end{aligned}
$$



9. Symmetric Encryption

9.7 Modes of Operation — CBC

- The correctness of the decryption follows from

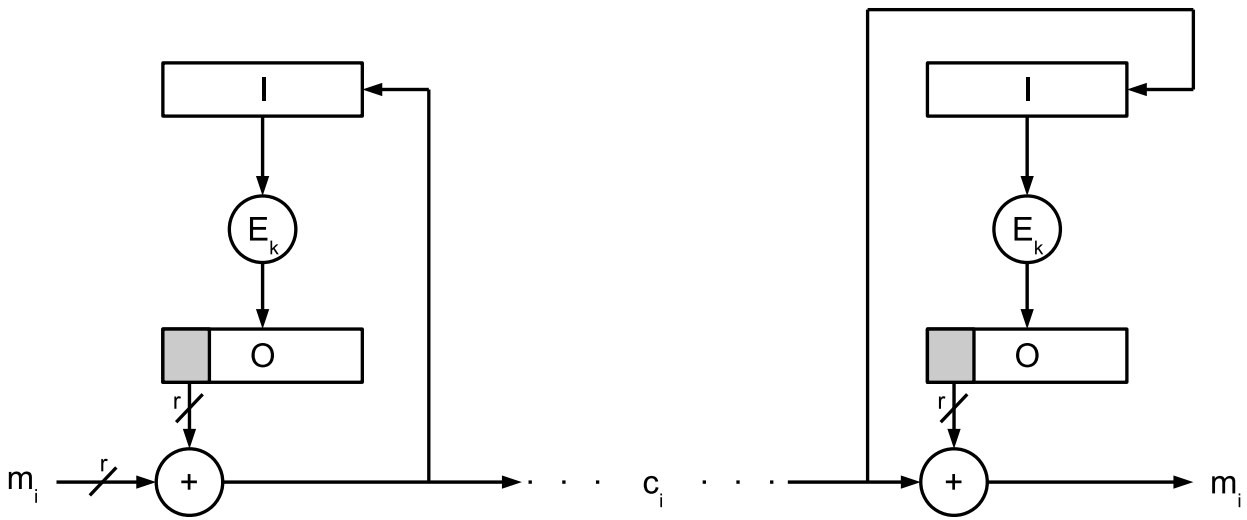
$$\begin{aligned} D_k(c_i) \oplus c_{i-1} &= D_k(E_k(m_i \oplus c_{i-1})) \oplus c_{i-1} \\ &= m_i \oplus c_{i-1} \oplus c_{i-1} \\ &= m_i \end{aligned}$$

- Due to the IV, there is a message expansion of one block (plus padding)
- Variants of "normal" CBC
  - CBC with ciphertext stealing
  - Propagating CBC (PCBC) — similar to Infinite Garble Extension (IGE) used in Telegram

9. Symmetric Encryption

9.7 Modes of Operation — CFB

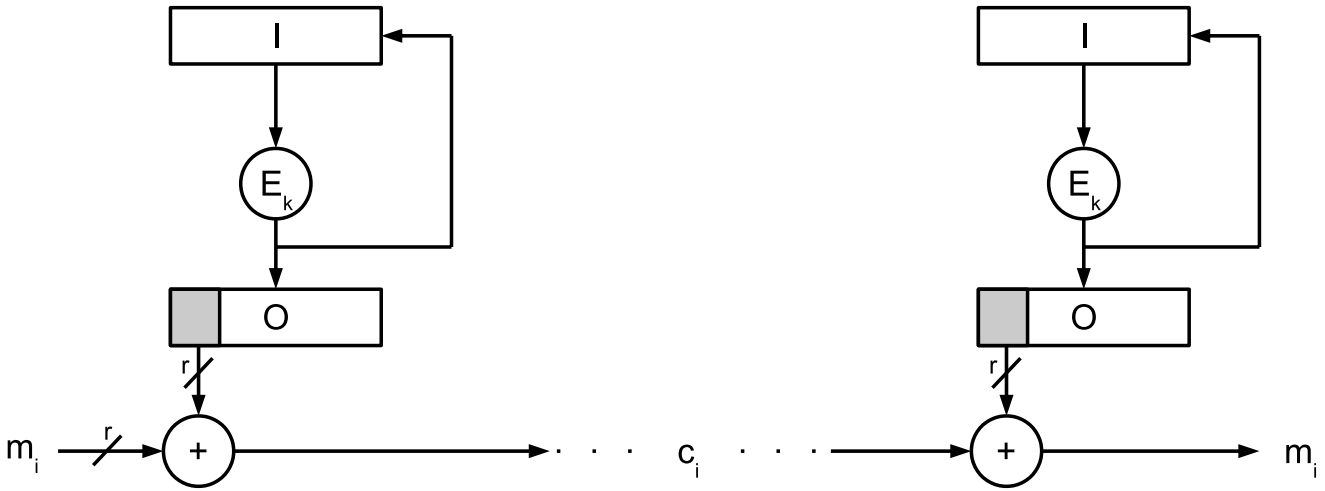
- CFB turns a block cipher into a (self-synchronizing) stream cipher



9. Symmetric Encryption

9.7 Modes of Operation — CFB

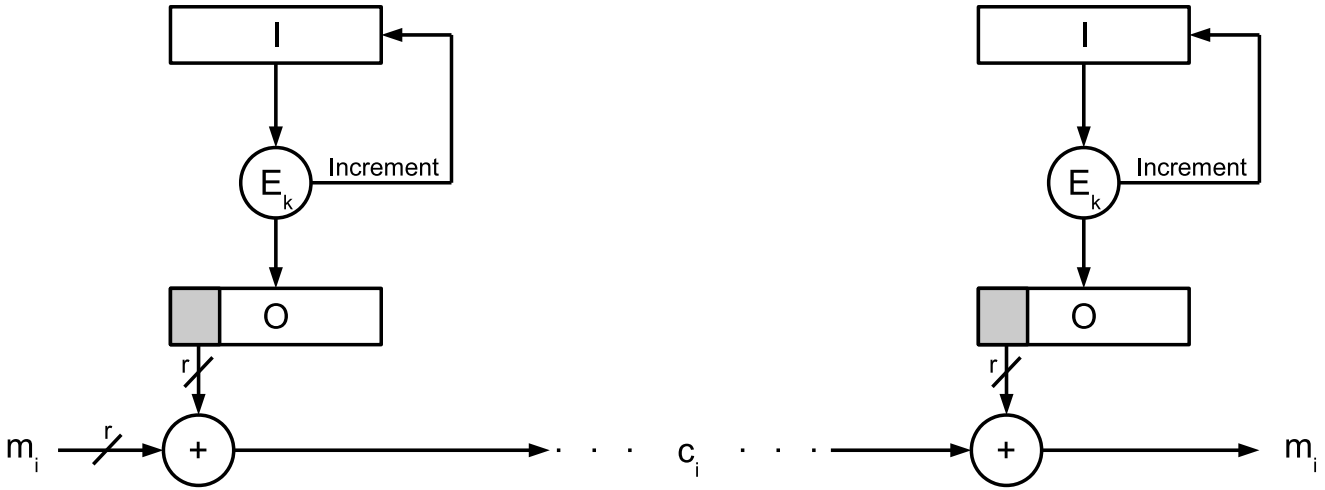
- OFB mode is similar to CFB, but it yields a synchronous stream cipher
- In contrast to CFB, the key stream can be precomputed



9. Symmetric Encryption

9.7 Modes of Operation — CTR

- CTR mode is similar to OFB, but it has a random access property that makes it suited for multiprocessor machines, where blocks can be encrypted and decrypted in parallel



9. Symmetric Encryption

9.8 Final Remarks

- DES (3DES) and RC4 are deprecated
- AES, Salsa20, and ChaCha20 are still in widespread use
- There are many other ciphers, e.g., the AES finalists, IDEA, FOX (IDEA-NXT), CAST, MISTY1, Camellia, SHACAL, ...
- All ciphers in use today look similar, i.e., they all employ a mixture of more or less complex functions that are iterated multiple times (i.e., in multiple rounds)
- The result is inherently hard to understand and (crypt)analyze

9. Symmetric Encryption

9.8 Final Remarks

- One may get the impression that it is simple to design and come up with a new cipher
- Unfortunately, this is not the case, and the design of a system that is secure and efficient is tricky
- Many ciphers had been proposed, implemented, and deployed, before a formerly unknown attack was discovered and applied to break them
- For example, the discovery of differential cryptanalysis brought the end to the Fast Data Encipherment Algorithm (FEAL) and many variants

9. Symmetric Encryption

9.8 Final Remarks

- Unless one enters the field of information-theoretical security, the level of security a cipher provides is difficult to determine
- Some cryptanalytical attacks are yet unknown and will be discovered in the future (probably)
- In this situation, it is simple to put in place and distribute rumors about possible weaknesses and vulnerabilities of particular ciphers
- This also applies to the influence and the cryptanalytical capabilities of NSAs

Questions and Answers

Thank you for your attention