Network Security

Chapter 4 – Symmetric Encryption

Cornelius Diekmann

With contributions by Benjamin Hof

Lehrstuhl für Netzarchitekturen und Netzdienste
Institut für Informatik
Technische Universität München

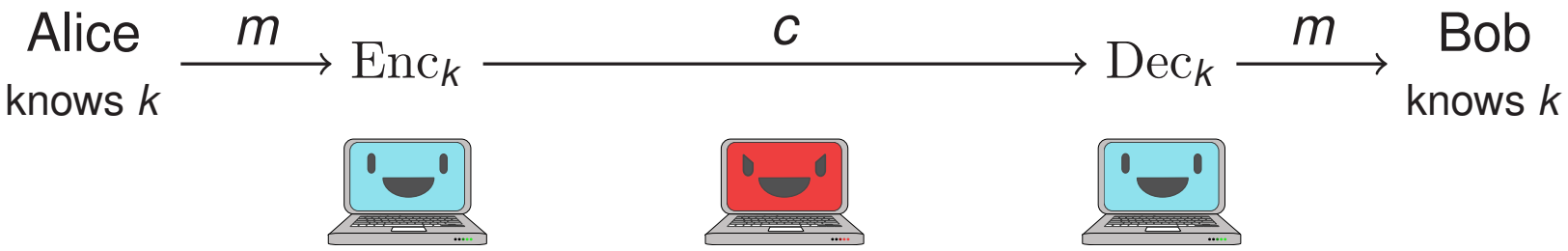
Version: October 29, 2015

Symmetric Encryption

Symmetric Encryption

- Alice and Bob share a secret key k
  - Implicit assumption: Only Alice and Bob know k
- The key is symmetric
  - Alice and Bob share the same k
  - The key is used to encrypt and decrypt
- Terminology
  - Plaintext m
    - The message itself
  - Ciphertext c
    - The encrypted plaintext
  - Encryption: $c = \mathrm{Enc}_k(m)$
  - Decryption: $m = \mathrm{Dec}_k(c)$
- Basic correctness requirement: $\mathrm{Dec}_k(\mathrm{Enc}_k(m)) = m$

Example



- $m =$ “This is network security”
- $k =$ 95 eb 50 0c 31 07 46 6f 88 8a f7 0b dd fb d7 64
- $c =$ ad 5c 66 d3 55 be 00 88 8c 82 41 d2 75 3d 93 da fe d0 12 20 ac c1 2c e6 64 60 b4 82 2c 87 03 b2
- $\mathrm{Enc} =$ AES-128-ECB

What security goals can we fulfill?

- Confidentiality?
  - Yes.
- Integrity?
  - No! An attacker could alter c.
- Authenticity?
  - No. Who are Alice and Bob anyway? Maybe Rogue-Alice is

Example for Enc and Dec: One-Time-Pad

One-Time-Pad: A Perfect Cipher

- Assumption: Alice and Bob share a perfectly random bitstream *otp*.
- $k = otp$
- $\mathrm{Enc}_{otp}(m) = m \oplus otp$
- $\mathrm{Dec}_{otp}(c) = c \oplus otp$
- Check:
  $\mathrm{Dec}_{otp}(\mathrm{Enc}_{otp}(m)) = \mathrm{Dec}_{otp}(m \oplus otp) = (m \oplus otp) \oplus otp = m$
- Requirements:
  - Key must have same size as message.
  - Key must only be used once.

Note: '$\cdot \oplus \cdot$' denotes XOR

Security of Ciphers

Kerckhoff's principle

The cipher method must not be required to be secret, and it must be able to fall into the hands of the enemy without inconvenience.

- In other words:
  - The cipher (encryption algorithm) is public.
  - Only the key is secret.

Examples of secure real-world ciphers

- AES
- 3DES
- ChaCha20
- One-Time-Pad

- Why can we trust them?
  - They have been publicly reviewed,
  - analyzed by cryptographers,
  - and standardized.
  - Well-tested implementations are available in your library

- Using them securely:
  1. RTFM
  2. keep the key secret (Kerckhoff's principle)

Repetition: Dos and Don'ts

- Do
  - Do use standardized ciphers from your library
  - Be aware of the dangers
    - Unlikely: A well-established cipher is broken or backdoored
    - Likely: Wrong usage of the cipher compromises security (RTFM)!
- Don't
  - Don't implement your own cipher. It will be broken, I guarantee!
  - Don't claim "*it's encrypted, it is secure*". Forgetting integrity and authenticity may be worse than any information leakage!
  - Don't forget about key management.

Attacking Symmetric Ciphers

Attacking Symmetric Ciphers

- Goal: given c, learn something about m
- Note: if something about k can be learned, the attack is successful. Why?
- Attack Scenarios:
  - Ciphertext-only-attack
    - Attcker knows c
  - Known-plaintext attack
    - For a fixed k, the attacker got a pair (m, c) and tries to learn something about *other* ciphertexts
  - Chosen-plaintext and chosen-ciphertext attack.
    - similar to previous attack, but attacker can chose m or c freely
- Examples in networks
  - passively sniffing attacker: usually ciphertext-only
  - attacking a server: chosen-plaintext
  - replaying eavesdropped modified messages: chosen-ciphertext

Security of Ciphers

Disclaimer: hand-waving idea. This is not a cryptography course.

- A cipher is secure if the best known attack is brute-forcing all keys.
- Brute-Force: exhaustively testing all keys
- Good keysize (symmetric cipher): 128 bit
  - A 10 Ghz CPU with 1 encryption operation per cycle
  - needs about 10^{22} years to brute-force the whole key space.
  - On average, only half of the possible keys must be tried, ...
  - only $5 \cdot 10^{21}$ years necessary

Example: Security of One-Time-Pad

One-Time-Pad: A Perfect Cipher

- c of $\mathrm{length}(c)$ can be decrypted to any m of length $\mathrm{length}(c)$
- Only knowledge of k reveals the right m
- OTP is a *perfect* cipher
- Attack scenarios in details
  - Ciphertext-only: No attack possible; any possible plaintext can be generated with the ciphertext.
  - Pairs of c and m don't help:
    The *otp* can be calculated, but this *otp* won't be reused!
  - Any statistical attack: due to *otp*, the ciphertext is perfectly random!

One-Time-Pad: Drawbacks

- Necessary key length in bits: $\mathrm{length}(k) = \mathrm{length}(m)$
- k must not be reused
- Wish list for practical ciphers
  - $\mathrm{length}(k) \ll \mathrm{length}(m)$
  - Key of fixed length, e.g. 128 bit
  - Key reusable for several messages
  - Unavoidable implication (for $\mathrm{length}(m) \gg \mathrm{length}(k)$):
    - Brute-forcing: $2^{\mathrm{length(k)}}$ instead of $2^{\mathrm{length(c)}}$ for otp.
    - Ciphertext-only attack succeeds w.h.p. when a k is found which decrypts c to an 'intelligible' m.
    - If m is not perfectly random, c cannot be perfectly random
  - Cipher is still secure

Example: An Insecure Cipher

Example: iCry – insecure cryptographic cipher

- $k \in \mathbb{B}^4$ key of length 4 bit
- Split m into blocks of 4 bit each: $m = m_1\ m_2\ m_3 \ldots$
- Encrypt each block individually with $\oplus$
- $\mathrm{Enc}_k(m_i) = m \oplus k = \mathrm{Dec}_k(c_i)$
- Example: encrypting “L”
  - $m =$ `ord('L') = 0x4c` = $0100_b\ 1100_b$
  - $k = 1010_b$
  - $c =$ `0xe6` (not an ASCII char)

	m_1:0100	m_2:1100
$\oplus$	k:1010	k:1010
	c_1:1110	c_2:0110

Example: Attacking iCry

- Known-plaintext attack
  - Attacker knows: $(m, c) = (0100_b\ 1100_b,\ \ 1110_b\ 0110_b)$
  - Attacker can compute k

    $k = 0100_b \oplus 1110_b = 1010_b \quad \text{or} \quad k = 1100_b \oplus 0110_b = 1010_b$
  - Attacker can now read all future messages encrypted with this k

Example: Attacking iCry

- Ciphertext-only attack: Attacker knows: $c = 1110_b\ 0110_b$

k	$m = \mathrm{Dec}_k(c)$	ASCII value
0000	11100110	[not an ASCII char]
0001	11110111	[not an ASCII char]
0010	11000100	[not an ASCII char]
0011	11010101	[not an ASCII char]
0100	10100010	[not an ASCII char]
0101	10110011	[not an ASCII char]
0110	10000000	[not an ASCII char]
0111	10010001	[not an ASCII char]
1000	01101110	n
1001	01111111	[non-printable ASCII char]
1010	01001100	L
1011	01011101	]
1100	00101010	*
1101	00111011	;
1110	00001000	[non-printable ASCII char]
1111	00011001	[non-printable ASCII char]

- Attacker brute-forces the small key space
- Intelligible decryptions: ‘n’ and ‘L’
- Possible keys:

  1000_b or 1010_b
- Attacker needs more ciphertext to improve the guess of the correct key
- (because k is reused)

Block and Stream Ciphers

Block and Stream Cipher

- Assumes: shared symmetric k of fixed length
- Block cipher
  - Encrypts and decrypts inputs of length n to outputs of length n
  - Block length n
  - Examples: AES, 3DES
- Stream cipher
  - Generates a random bitstream, called *keystream*
  - $c = keystream \oplus m$
  - Examples: ChaCha20, RC4 (broken!)

Example: Block Cipher AES-128

- AES-128
  - blocks size: 128 bit (16 bytes)
  - key size: 128 bit
- $m =$ “This is network.”
- $\texttt{len}(m) = 16$ bytes
- $k =$ 128 truly random bits
- $\mathrm{Enc}_k(m) =$ 2d 3c ab 1b a0 80 77 ec e8 1d 56 0d 09 2b f6 77

Example: Some Stream Cipher

- $m =$ “HELLO” = `48 45 4c 4c 4f`
- $k =$ `streamcipher.get_keystream_bytes(5)` $=$ `12 a7 f9 07 55`
- $\mathrm{Enc}_k(m) = k \oplus m =$ `5a e2 b5 4b 1a`

	0100 1000	0100 0101	0100 1100	0100 1100	0100 1111
$\oplus$	0001 0010	1010 0111	1111 1001	0000 0111	0101 0101
	0101 1010	1110 0010	1011 0101	0100 1011	1000 1010

Interlude: Which Crypto Cipher should I use?

- Probably AES

- Reasons to use AES
  - Fast: 200 MBit/s in software and $>$ 2 GB/s with Intel AES-NI
  - Hardware implementations for embedded devices available
  - A well-tested implementation is available in your library
  - Secure (attacks exist, but AES is practically secure)
  - AES seems to be the best we have, and it is among the most researched algorithms

Modes of Encryption

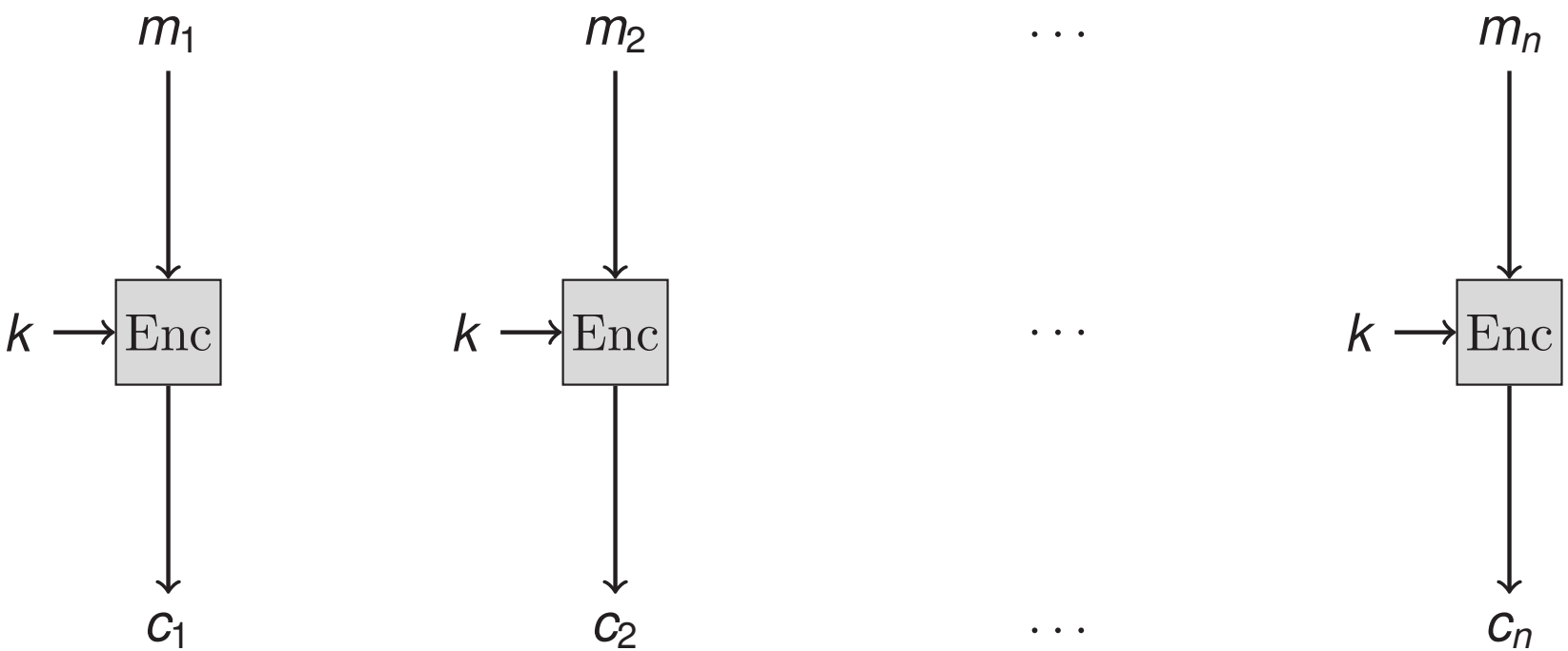
Modes of Encryption: Motivation

- Block ciphers handle messages of length x
- Problem: $\mathrm{length}(m) \gg x$
- Solution: Modes of Encryption

- We split m into blocks m_i where $\mathrm{length}(m_i) = x$
- $m = m_1 \; m_2 \; \ldots \; m_n$
- if $\mathrm{length}(m)$ is not a multiple of x, the last block is filled up
- Technical Term: padding

Electronic Code Book Mode – ECB

- $c_i = \mathrm{Enc}_k(m_i)$



ECB – Example

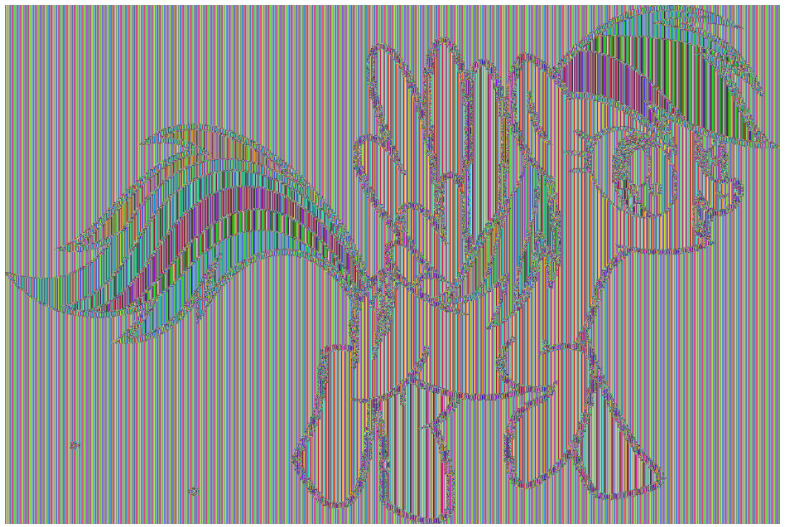
- $m =$ “This is network.This is network.Security”
- $\mathrm{Enc} =$ AES-128, mode $=$ ECB
- $c =$

```
2d 3c ab 1b a0 80 77 ec e8 1d 56 0d 09 2b f6 77
2d 3c ab 1b a0 80 77 ec e8 1d 56 0d 09 2b f6 77
16 ea 2c 19 97 e7 40 db 06 a0 35 93 49 5c 37 0b
```

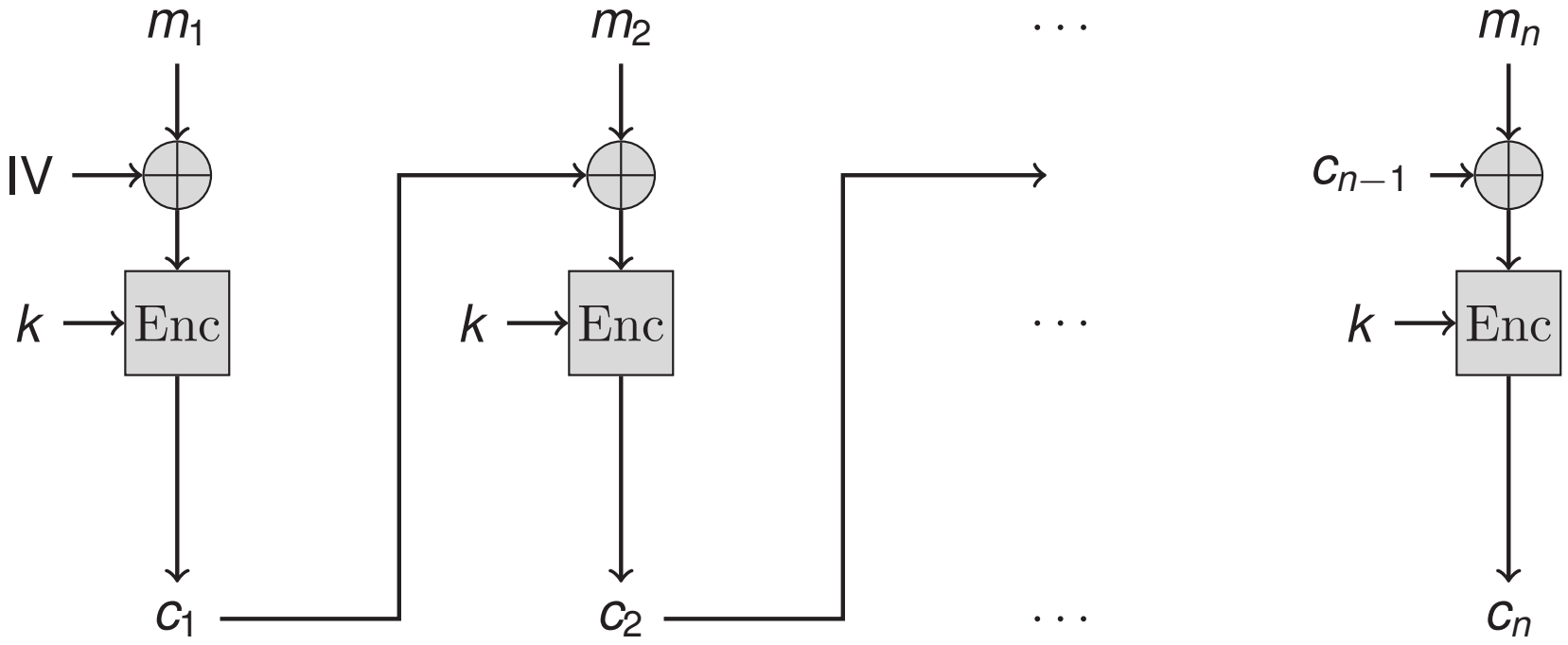
- Why are line 1 and line 2 identical?
- $m_1 =$ “This is network.”
- $m_2 =$ “This is network.”
- $m_3 =$ “Security” + padding

ECB – Drawback

- Identical plaintext blocks are encrypted to identical ciphertext!



Cipher Block Chaining Mode – CBC



CBC – Discussion

- CBC Encrypt: $c_i = \mathrm{Enc}_k(c_{i-1} \oplus m_i)$
- Why the $\oplus$ with the previous block?
  - Identical plaintext blocks are encrypted to non-identical ciphertext
- $c_0 = \mathrm{IV}$
- What is the use of the IV (initialization vector)?
  - Completely identical messages are encrypted to non-identical ciphertexts
- IV may be public
- IV must be fresh

CBC – Example

- Sending m encrypted over UDP, using CBC.
- m is split into blocks for the block cipher.
- $m = m_1\ m_2\ m_3\ m_4\ m_5\ m_6$
- m is split over two UDP packets.
- A new and random IV is put in clear at the beginning of the payload of every packet.

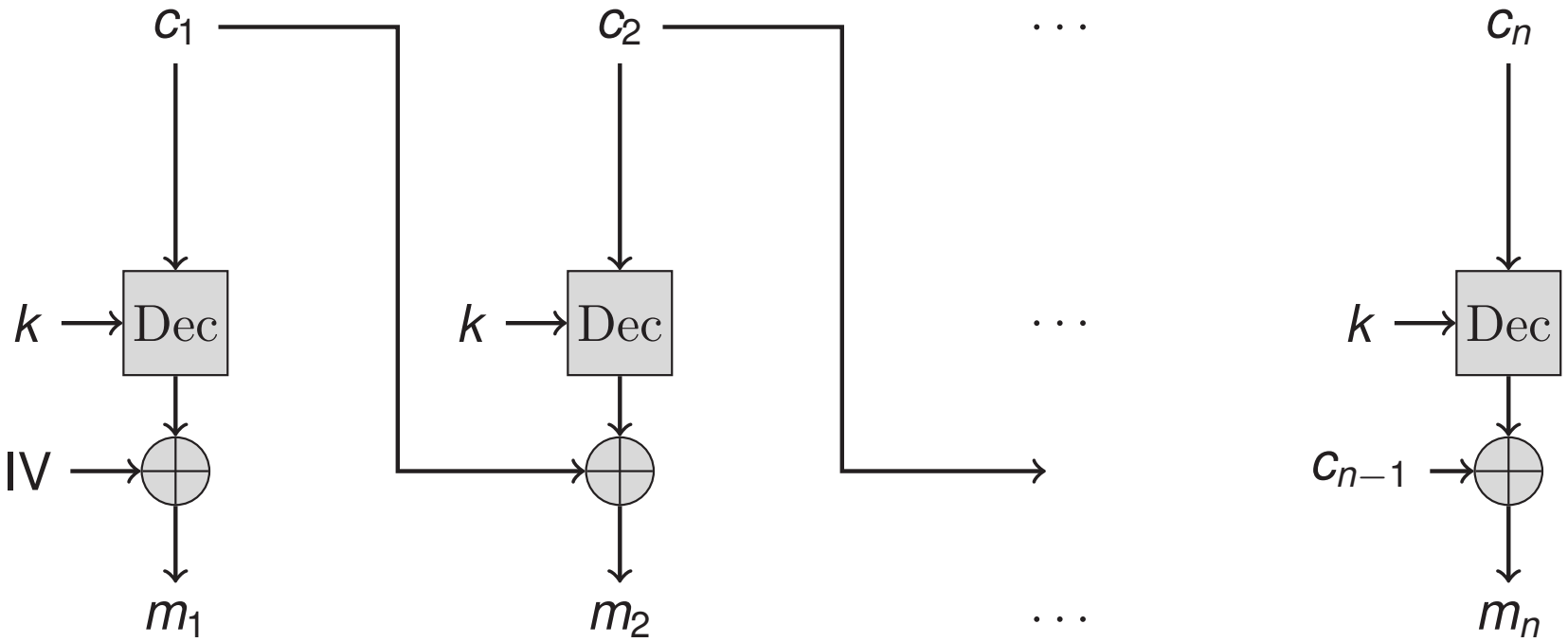
IP header
UDP header
IV_1
c_1
c_2
c_3

IP header
UDP header
IV_2
c_4
c_5
c_6

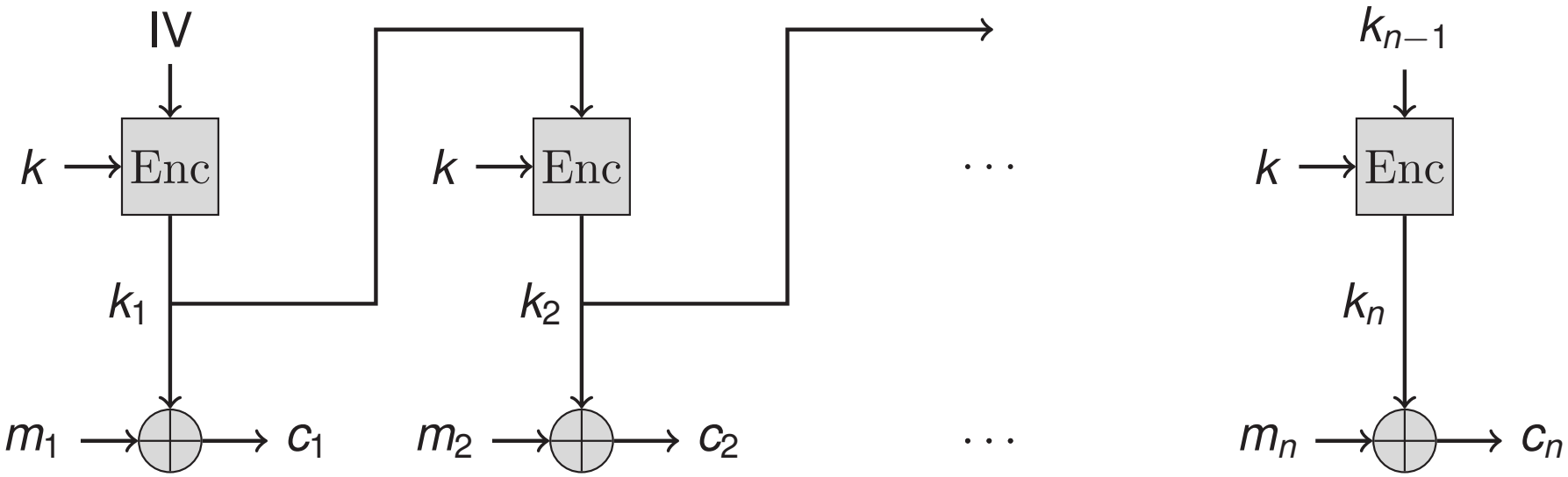
CBC – Decrypt

- CBC Encrypt: $c_i = \mathrm{Enc}_k(c_{i-1} \oplus m_i)$
- $c_0 = \mathrm{IV}$
- Let's do the math:
- $c_i = \mathrm{Enc}_k(c_{i-1} \oplus m_i)$
- $\mathrm{Dec}_k(c_i) = \mathrm{Dec}_k(\mathrm{Enc}_k(c_{i-1} \oplus m_i))$
- $\mathrm{Dec}_k(c_i) = c_{i-1} \oplus m_i$
- $\mathrm{Dec}_k(c_i) \oplus c_{i-1} = m_i$
- CBC-Decrypt: $m_i = c_{i-1} \oplus \mathrm{Dec}_k(c_i)$

CBC Decrypt

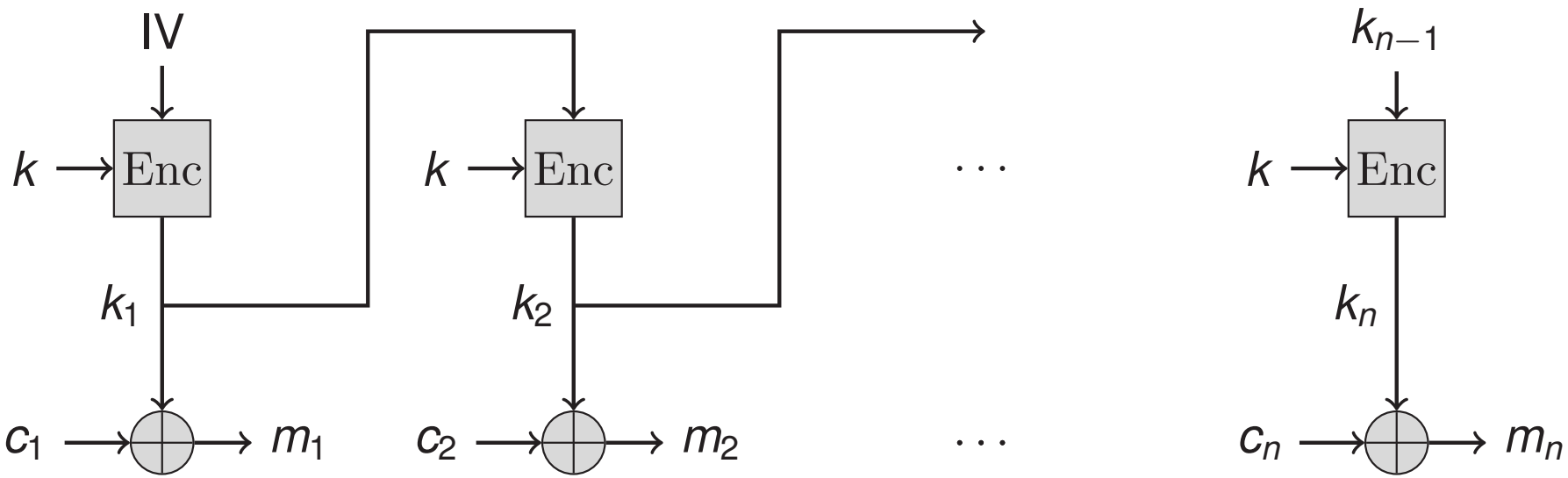


Output Feedback Mode – OFB



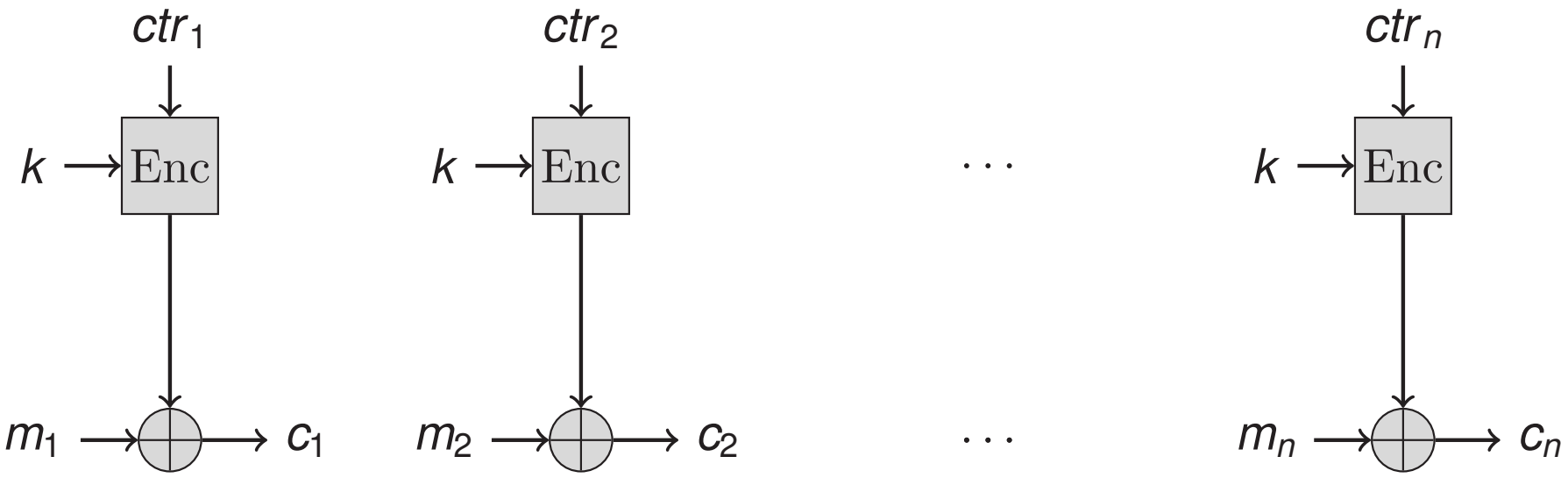
- Transforms a block cipher into a stream cipher.
- IV may be public but must be fresh.

OFB – Decrypt



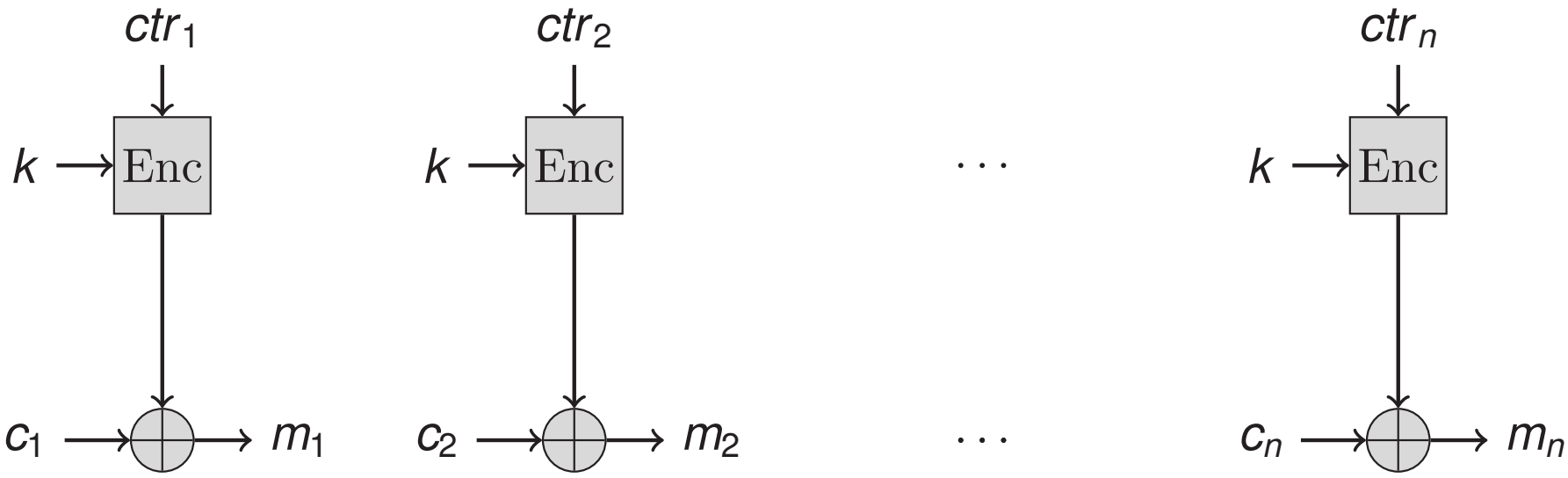
Counter Mode – CTR

- $ctr_i = \mathrm{IV} \parallel i$



- Transforms a block cipher into a stream cipher.
- IV may be public but must be fresh.

CTR Decrypt



Literature

- Jonathan Katz and Yehuda Lindell, *Introduction to Modern Cryptography*, 2nd edition, CRC Press, 2015 $\Longleftarrow$ recommended
- Filippo Valsorda, *The ECB Penguin*, PyTux Blog, 2013, `https://filippo.io/the-ecb-penguin/`
- Günter Schäfer, *Security in Fixed and Wireless Networks: An Introduction to Securing Data Communications*, Wiley, 2004
- Günter Schäfer, *Netzsicherheit*, dpunkt, 2003