B.b) Block Ciphers

B.24 Definition

- An encryption algorithm

  Enc: $\{0,1\}^n \times K \rightarrow \{0,1\}^n$

  is called a *block cipher*.
- The positive integer n denotes the *block size* (*block length*).

B.25 Remark and Convention

- The encryption transformation and the decryption transformations are permutations over $\{0,1\}^n$.
- Usually, block ciphers are symmetric. In this course we only treat symmetric block ciphers.
- To prevent elementary attacks (e.g., elementary frequency analysis; cf. the attack on the improved variant of Cesar's cipher) the block size n should not be too small.
- Typical block sizes are n = 64 and n = 128.
- For many widespread block ciphers $K = \{0,1\}^m$.

B.26 ECB mode (Electronic Code Book mode)

Goal: Encrypt a bit string $b_1,b_2,\ldots,b_M$

- 1st Step: Padding
  - If M is not a multiple of the block length n append some bits (padding bits) $b_{M+1},\ldots,b_{nt}$ to this bit string
  - Partition $b_1,b_2,\ldots,b_{nt}$ into t non-overlapping blocks $p_1,p_2,\ldots,p_t$

    (More precisely, $p_1=(b_1,b_2,\ldots,b_n)$, $p_2=(b_{n+1},b_{n+2},\ldots,b_{2n})$, $\ldots \in \{0,1\}^n$.)

B.26 ECB mode (continued)

Note:

(i) The receiver must be able to determine the padding bits unambiguously.

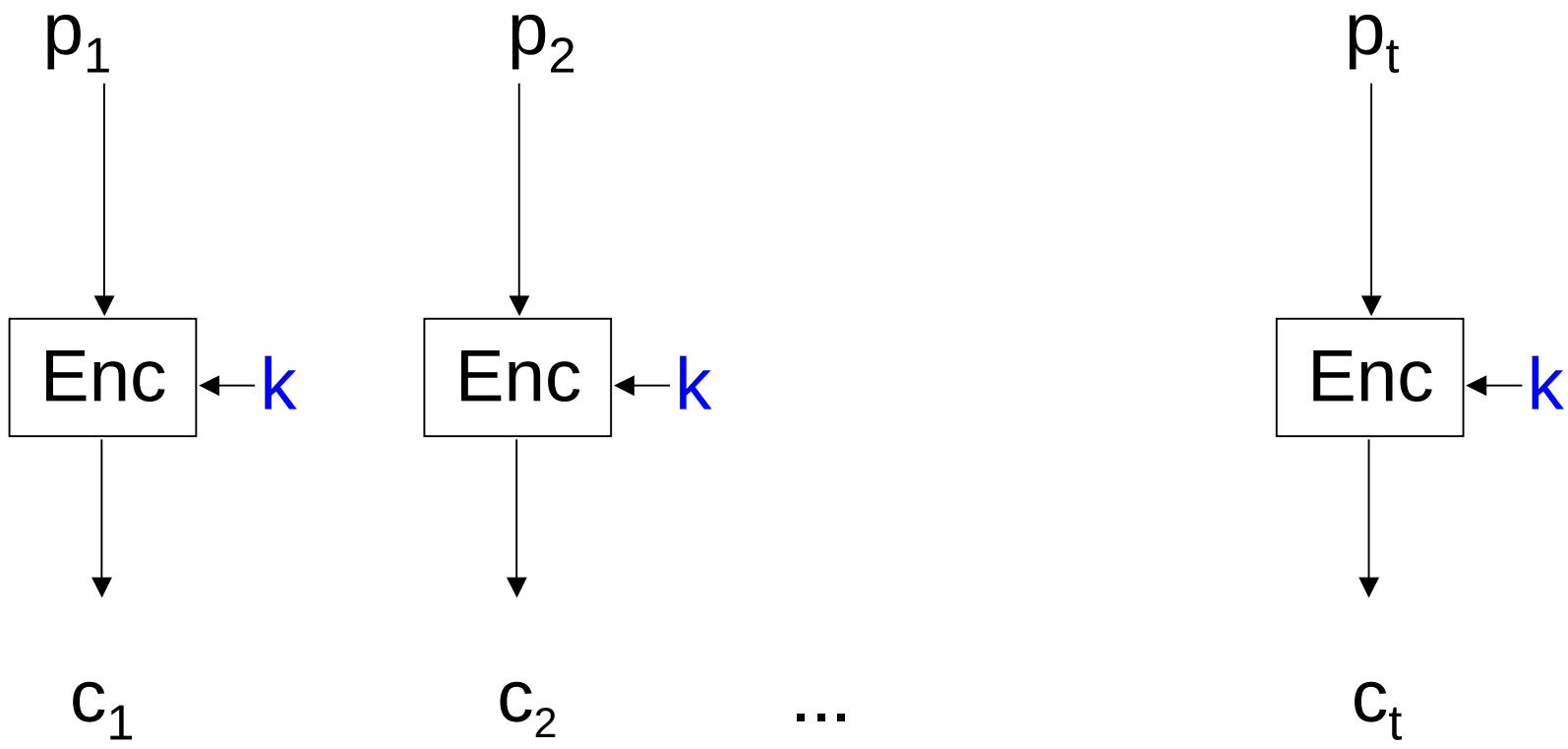
(ii) Example (padding): $b_{M+1} = 1$, $b_{M+2} = \ldots = b_{nt} = 0$

(iii) In various padding schemes padding bits are always appended, even if M is a multiple of n.

(iv) Depending on the concrete application non-random padding bits may allow the receiver to perform an integrity check after decryption.

B.26 ECB mode (continued)

- 2nd Step: Encrypt the plaintext blocks $p_1,\ldots,p_t$



$c_j = Enc(p_j,k)$

B.26 ECB mode (continued)

The receiver receives the encrypted message $c_1,\ldots,c_t$

Decryption:

1st Step: The receiver computes $p_j = Dec(c_j,k)$ for $j = 1,2,\ldots,t$

2nd Step (integrity check if the application permits; optional): The receiver checks whether the format of the decrypted message is syntactically o.k.

3rd Step: The receiver removes the padding bits, obtaining the original plaintext message.

B.27 ECB mode: Significant Properties

- In the ECB mode identical plaintext blocks are encrypted to identical ciphertext blocks. This property allows elementary attacks. Note that
  - Reordering the ciphertext blocks yields reordered plaintext blocks after decryption.
  - Doubling a particular ciphertext block gives a doubled plaintext block after decryption.
  - If the plaintext blocks represent the colour of (one or several) pixels in a coarse graphics an adversary might be able to recognize its main contours without breaking the cipher.

Details: blackboard Example: Exercises

B.27 (continued)

- Error propagation: Bit errors in a single ciphertext block only affect the decryption of this particular block.
- Note: For a strong block cipher even a single bit flip causes about $n / 2$ bit errors in the decrypted block in average.

B.28 ECB mode: Fields of Application

- Encryption of 1-block messages
- Occasionally: Encryption of short messages in smart card communication (Note: Smart card applications often determine the number of plaintext blocks and the syntax within these blocks, leaving little variability. This prevents attacks as mentioned in B.27).
- Key derivation

B.29 Chip-based Electronic Purse Systems

Simplified description

Involved parties

- Electronic purses (smart cards that store units that represent money)
- merchant terminals (often supported by smart cards that perform the sensitive operations)
- background system

Functionality: cashless payment (offline transactions)

B.29 (continued)

Cashless Payment:

- The merchant terminal displays the price *pr* and transmits an accordant message to the electronic purse.
- The electronic purse
  - reduces its balance by *pr* units
  - transmits a payment record to the merchant terminal, confirming this action
- The merchant terminal
  - stores this payment record (offline system)
  - submits the collected records periodically to the background system
- The background system books money to the merchant's account

B.29 (continued)

Loading the electronic purse:

- The holder of the electronic purse pays cash or with book money *x* €
- The background system increases the balance of the electronic purse by *x* units (1 unit = 1 €).

B.29 (continued)

Security requirements:

- The merchant terminal must only accept payment records from authentic electronic purses. (Otherwise an adversary could use duplicates of electronic purses for which he could increase the balance himself.)
- The merchant must not be able to generate new payment records or submit authentic payment records more than once.
- Only the background system shall be able to load electronic purses.

B.30 Example

Scenario: A smart card terminal shares a key k with each smart card that supports a particular application, e.g. that belongs to a particular electronic payment system.

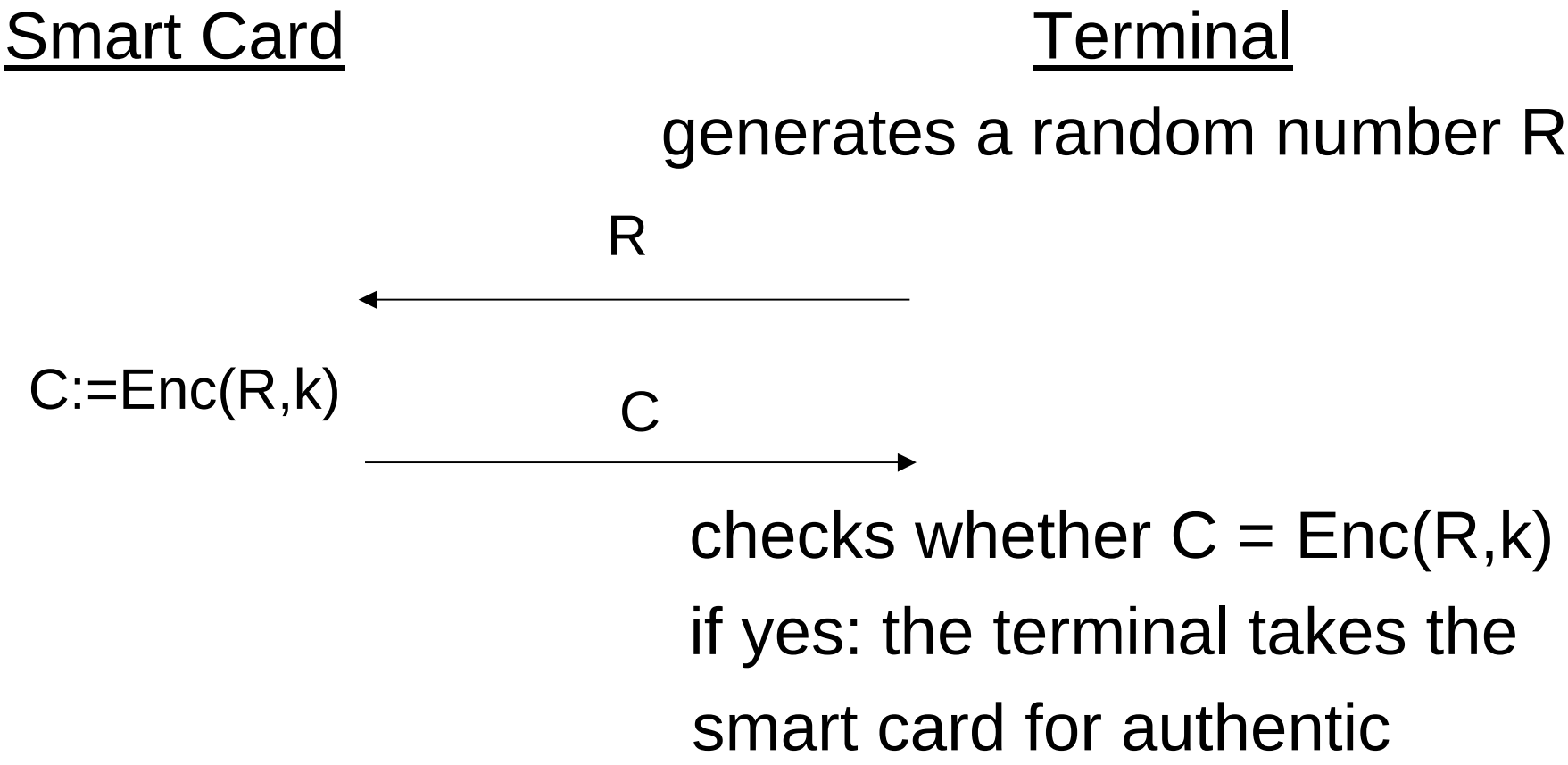
Goal: The smart card shall convince the terminal that it is authentic.

Straight-forward solution: The smart card transmits k as a password. The terminal believes a smart card to be authentic if it sends k.

Drawback of this solution: An adversary might monitor exchanged messages, learning k.

B.30 (continued)

Better solution: Dynamic Authentication



B.30 (continued)

- This is the most elementary *challenge-response protocol.*
- Monitoring messages from authentic smart cards does not help to pass the challenge response protocol since the challenge R changes from run to run.
- Monitoring old messages only provides (plaintext, ciphertext) pairs for known plaintext attacks on Enc.

B.31 Example

For sensitive applications smart cards usually have individual keys so that a successful attack on one smart card does not compromise the all the others.

Goal: Perform the challenge-response-protocol from B.30 with card-individual keys.

Straight-forward solution: The terminal stores the individual keys of all smart cards. This solution is appropriate for

- online systems with one key list on a central server
- offline systems with a small number of smart cards

Drawback of this solution: For widespread offline applications thousands or even millions of keys had to be stored. Moreover, the list had to be updated whenever a new smart card is issued.

B.31 (continued)

More efficient solution: Each smart card has an individual card number C_Nr and an individual key k_C . The terminals have a master key k_M.

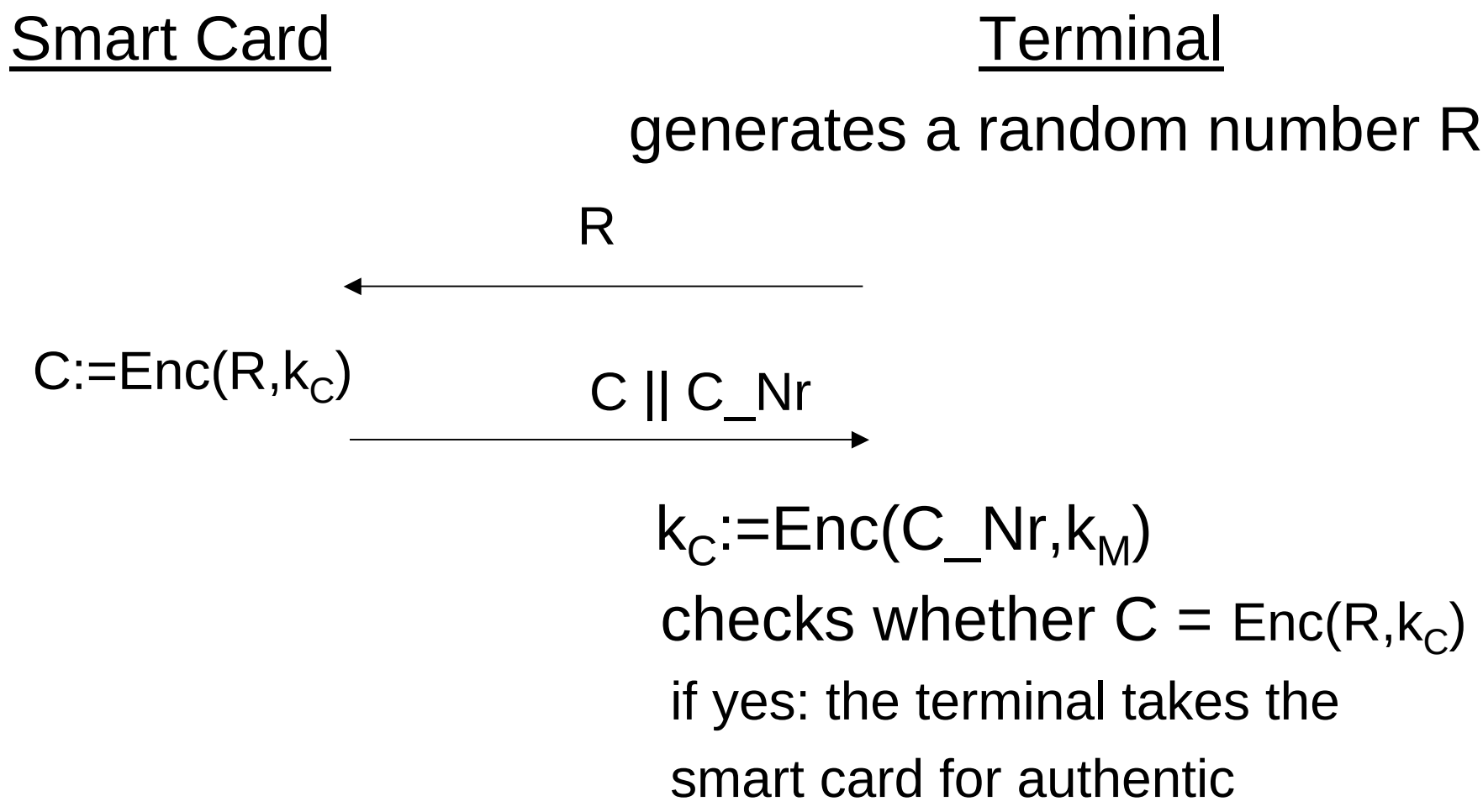
Key derivation: $k_C = Enc(C\text{-}Nr, k_M)$

Note: (I) Alternatively, k_C might be a given by a function value of the right-hand side.

(ii) There also exist key derivation mechanisms that apply one-way functions (cf. Chapter C).

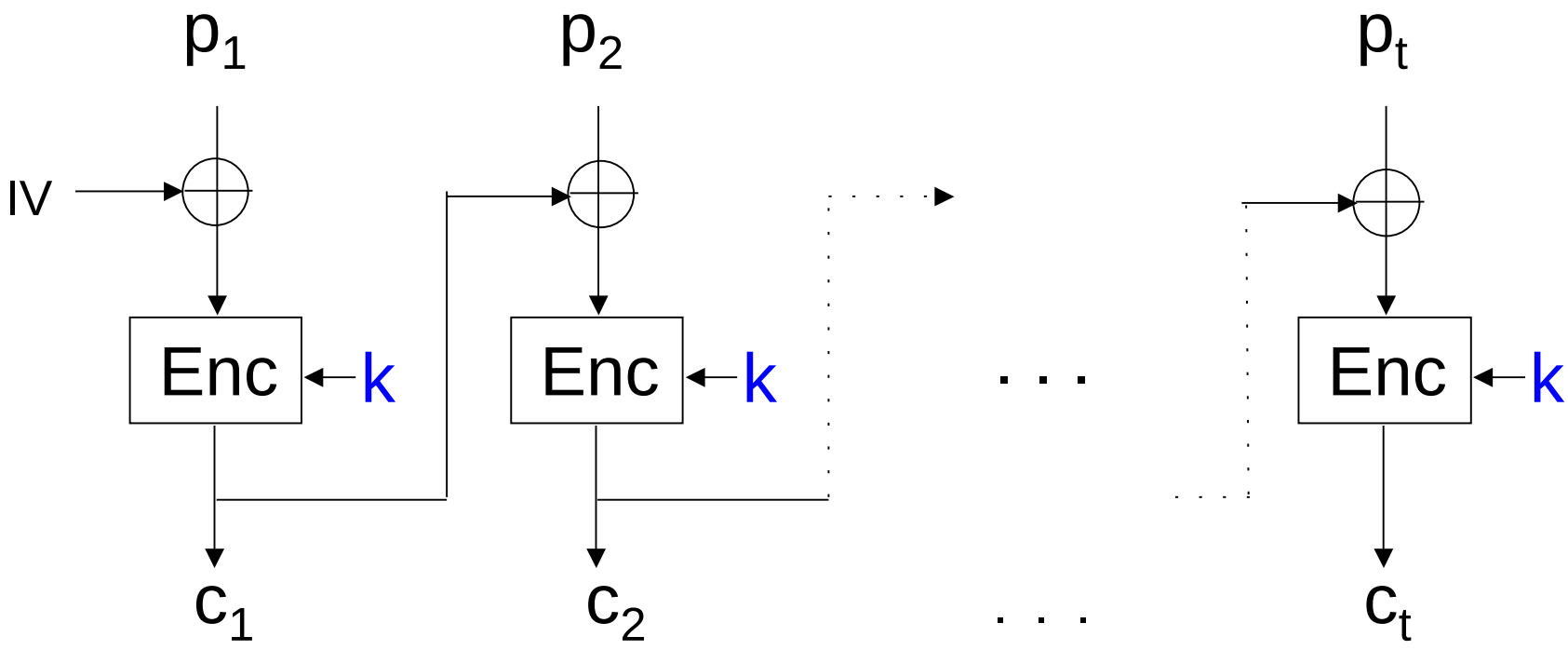
The challenge response protocol from Example B.30 now reads as follows:

B.31 (continued)



B.32 CBC mode (Cipher Block Chaining mode)

- 1nd Step: Padding (as in ECB mode)
- 2nd Step: Encrypt the plaintext blocks $p_1,\ldots,p_t$



$$c_j = Enc(p_j \oplus c_{j-1}, k)$$

B.32 CBC mode (continued)

The receiver receives the encrypted message $c_1,\ldots,c_t$

Decryption:

1st Step: The receiver computes $p_j = Dec(c_j,k) \oplus c_{j-1}$ for j = 1,2,…,t with c_0 := IV

2nd Step (integrity check if the application permits; optional): The receiver checks whether the format of the decrypted message is syntactically correct.

3rd Step: The receiver removes the padding bits, obtaining the original plaintext message.

Note: Step 2 and Step 3 are the same as for the ECB mode

B.33 CBC mode: Significant Properties

- Different IVs cause different ciphertexts even for identical plaintexts and identical key. Varying the IV prevents *replay attacks* where an active adversary sends “old” authentic ciphertexts, which he has previously intercepted.
- The ciphertext block c_j depends on IV,p_1,…,p_j. Unlike in the ECB mode rearranging or doubling ciphertext blocks will presumably not give meaningful plaintext.
- Decryption may be parallelized.

B.33 (continued)

- Error propagation / error recovery: Since p_j only depends on c_{j-1} and c_j bit errors and even losses of blocks are compensated two blocks later.
- Often the IV does not need to be kept secret but its integrity must be ensured.

Note:

$p_1 = Dec(c_1,k) \oplus IV$ implies

$p_1^* := p_1 \oplus (IV \oplus IV^*) = Dec(c_1,k) \oplus IV^*$ for any IV*

B.34 Forging the IV (Attack)

Assumption: The adversary is able to read and to alter messages that are exchanged between the sender and the receiver.

Attack:

(i) The sender transmits $(c_1,\ldots,c_t; IV)$

(ii) The adversary alters the message to

$(c_1,\ldots,c_t; IV^*)$

(iii) The receiver decrypts the altered ciphertext and obtains $p_1^* := p_1 \oplus (IV \oplus IV^*), p_2,\ldots,p_t$

Example: Exercises

B.35 CBC mode: Fields of Application

- Encryption of long messages, e.g. applied in the SSL protocol
- MACs (Message Authentication Codes; cf. B.48)

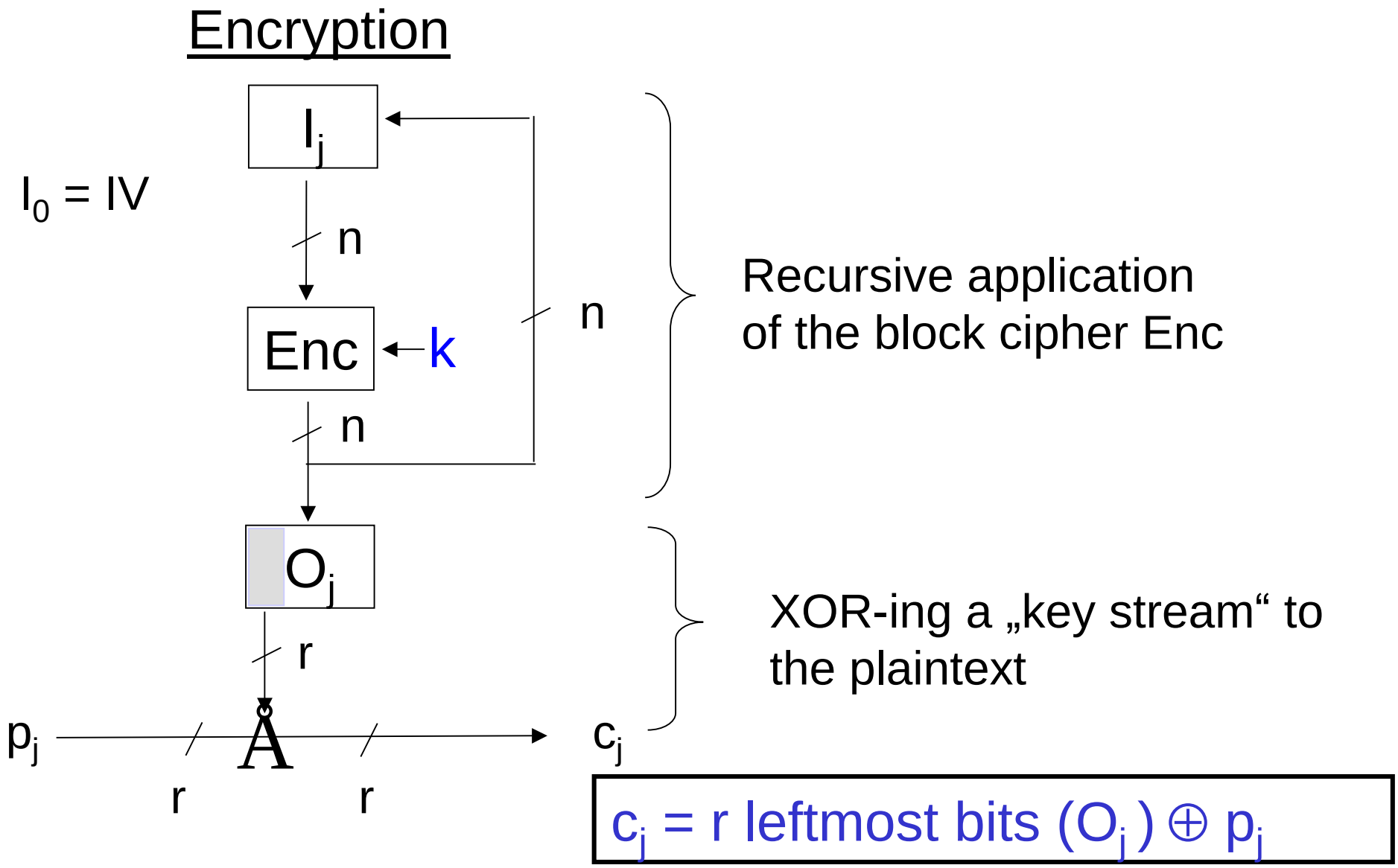
B.36 OFB mode (Output Feedback mode)

Encryption: Select an integer $r \leq n$

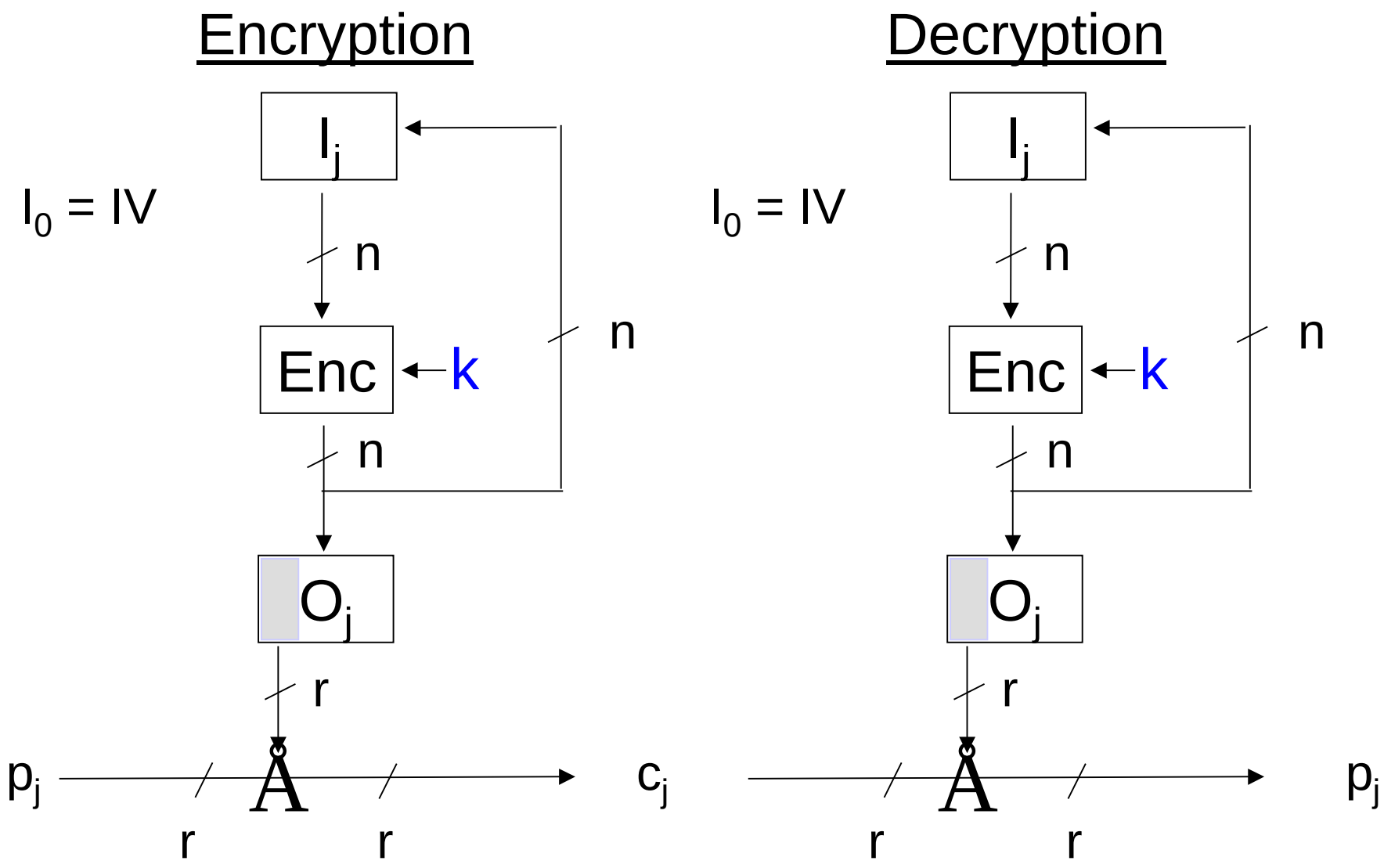
- Partition the plaintext into non-overlapping blocks $p_1,\ldots,p_s$ with $length(p_j)=r$ for $j<s$ and $length(p_s) \leq r$
- Encrypt these plaintext blocks

Note: In OFB mode padding is not necessary. If $length(p_s) < r$ then the $(r - length(p_s))$ rightmost bits of O_s (next slide) are neglected.

B.36 OFB mode (continued)



B.36 OFB mode (continued)



B.37 OFB mode: Significant Properties

- Encryption and decryption are identical operations.
- The OFB mode defines a *stream cipher* ($\rightarrow$ Section B.c). The output blocks O_1,O_2,…can be precomputed since they only depend on IV and k but neither on the plaintext nor on the ciphertext.
- Encryption and decryption of a plaintext block, resp. of a ciphertext block, only depends on the position of the block but not on its predecessors or successors.

B.37 (continued)

Error propagation / error recovery: Bit errors in c_j only affect the decryption of this particular block. A manipulation of the ciphertext should be easier than in CBC mode, for instance. Losses of blocks cannot be compensated.

B.37 (continued)

- Note: Identical IVs (with identical key k) generate identical sequences O_1, O_2, …
- Consequence: The IV must not be used twice in combination with the same key. In fact, since

$c_j \oplus c_j^* =$

$(r \text{ leftmost bits } (O_j) \oplus p_j) \oplus (r \text{ leftmost bits } (O_j) \oplus p_j^*) =$

$p_j \oplus p_j^*$

a ciphertext-only attack provides information on the plaintext (however, neither on IV nor on k)

B.38 OFB mode: Fields of Application

- real-time applications
- generation of pseudorandom numbers (= r left-most bits of O_1,O_2,…)

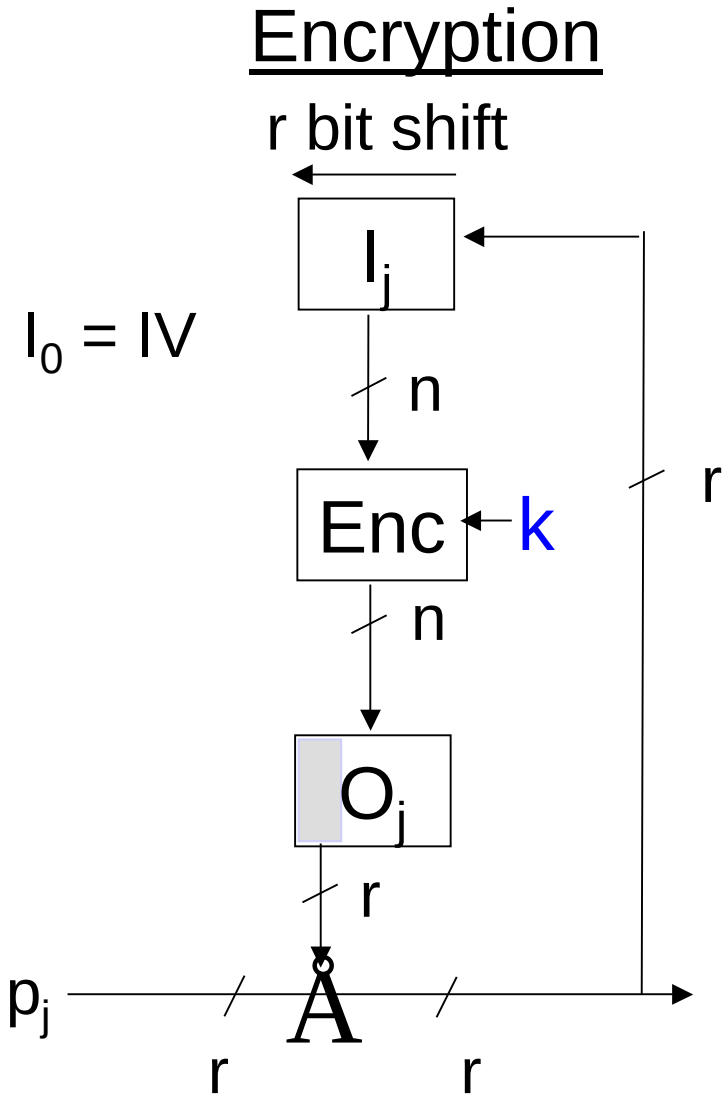
B.39 CFB mode (Cipher Feedback mode)

Encryption: Select an integer $r \leq n$

- Partition the plaintext into non-overlapping blocks $p_1,\ldots,p_s$ with $length(p_j)=r$ for $j<s$ and $length(p_s) \leq r$
- Encrypt these plaintext blocks

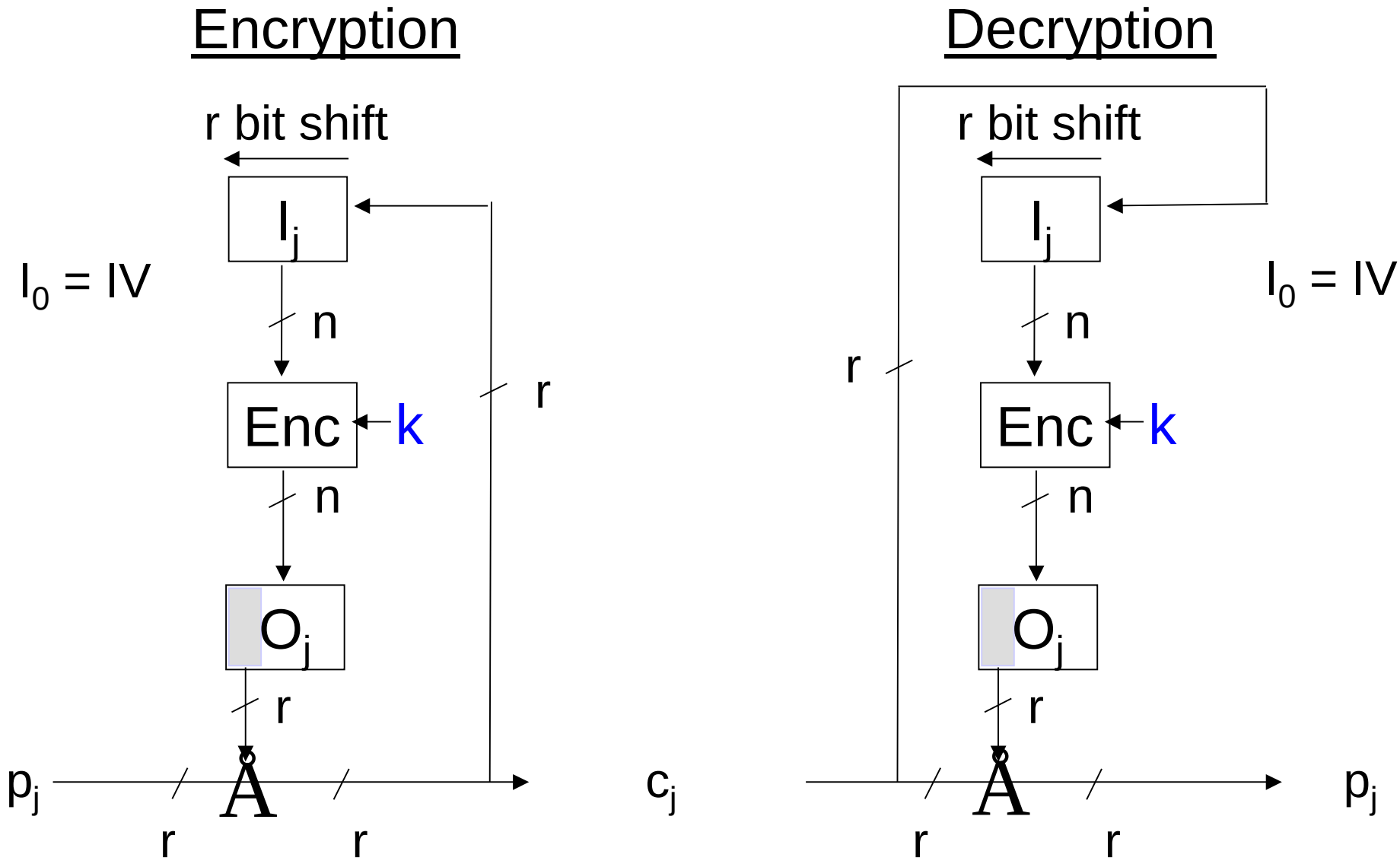
Note: As in OFB mode padding is not necessary.

B.39 CFB mode (continued)



c_j = r leftmost bits (O_j) $\oplus$ p_j

B.39 CFB mode (continued)



B.40 CFB mode: Significant Properties

- Encryption and decryption are identical operations.
- Error propagation / error recovery: Bit errors in c_j only affect the decryption of those cipherblocks c_m ($m > j$) for which c_j is part of I_m.

  Losses of ciphertext blocks influence only the decryption of the next ciphertext blocks since (for fixed key) O_j only depends on I_j. That is, the decryption works correctly for c_m if the lost ciphertext blocks are not part of I_m on the sender's side.
- The receiver may parallelize the computation of the key stream.

B.41 CFB mode: Fields of Application

- OpenPGP

B.42 CTR mode (Counter mode)

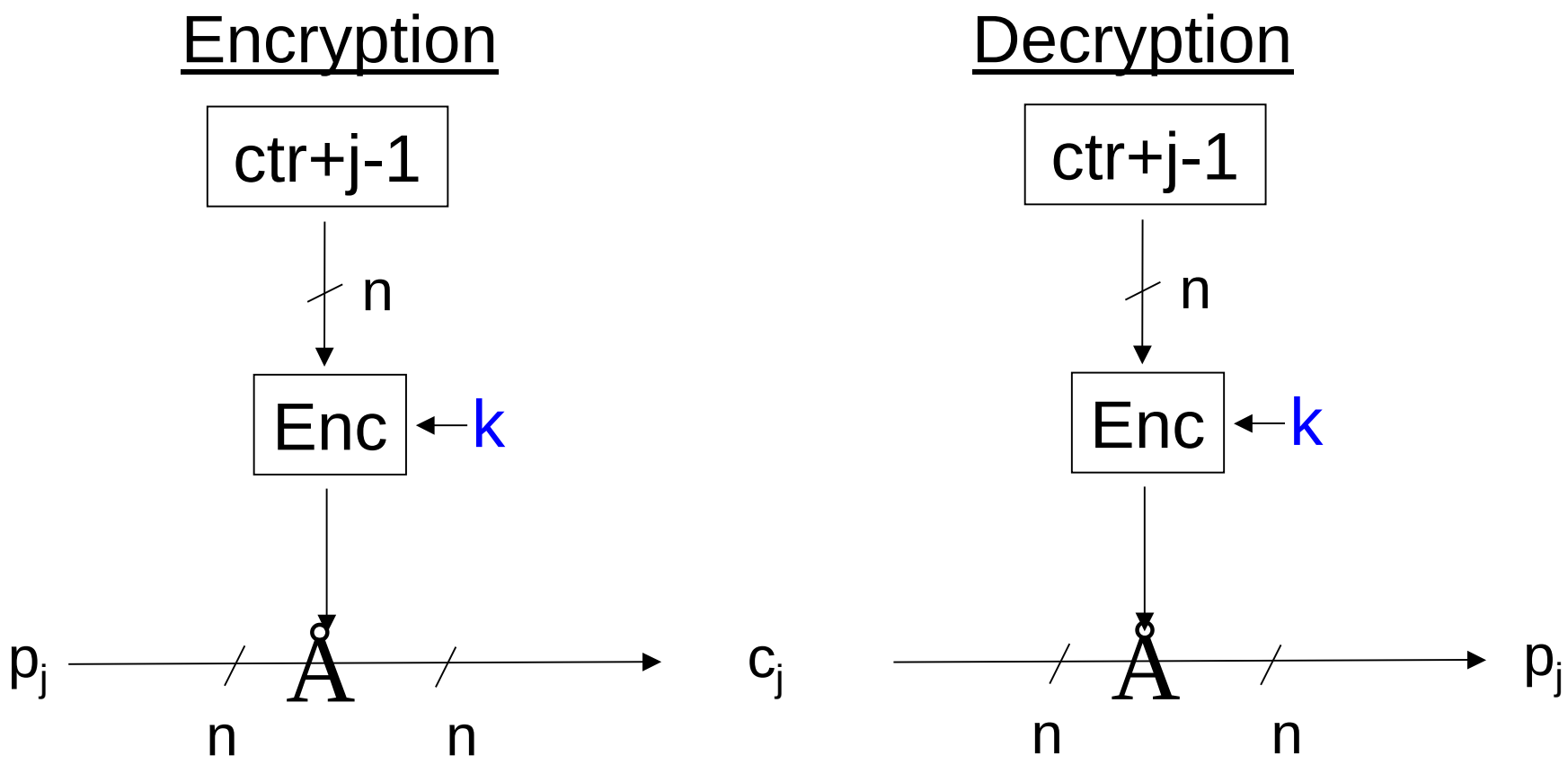
- 1nd Step: Select a counter value ctr (n bit value)
- 2nd Step: Encrypt the plaintext blocks $p_1,\ldots,p_t$

Note:

(i) As in OFB and CFB mode padding is not necessary (see B.36).

(ii) The ctr value corresponds to the IV in the OFB mode mode. In particular, the same ctr value should not be used twice. To be more precisely: the counter values (cf. the next slide) should not even overlap for different encryptions with the same key k.

B. 42 CTR mode (continued)



$$c_j = p_j \oplus Enc(ctr+j-1,k)$$

B.43 CTR mode: Significant Properties

- Encryption and decryption are identical operations.
- The CTR mode defines a stream cipher ($\rightarrow$ Section B.c). The key stream can be precomputed.
- Encryption and decryption of a plaintext block, resp. of a ciphertext block, only depends on the position of that block but not on its predecessors or successors.
- Encryption and decryption can be parallelized.
- Error propagation / error recovery: Bit errors in c_j only affect the decryption of this particular block. Losses of blocks cannot be compensated.

B.44 CTR mode: Fields of Application

- disk encryption

B.45 Remark

Note: Besides the ECB, CBC, OFB, CFB and the CTR mode several other modes of operation have intensively been discussed in the literature. In this course we will not treat further modes.

The designer of a cryptosystem should decide for that mode of operation that is most appropriate for the intended application(s).

B.46 Example (cf. Example B.31)

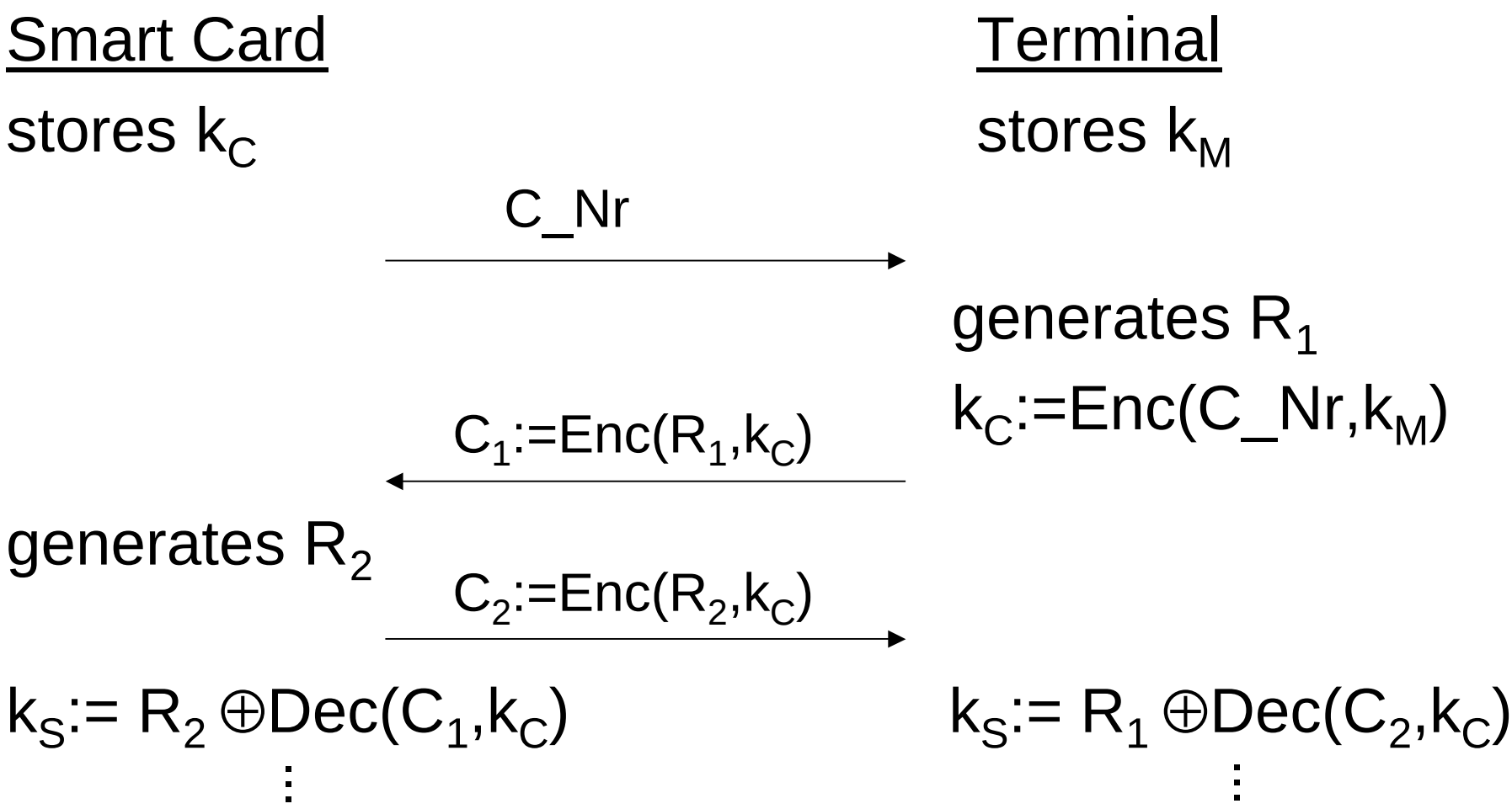
Challenge-response protocol with a smart card and a terminal. As in B.31 the terminal has a master key.

Goals:

- mutual authentication (i.e. both the smart card and the terminal prove that they are authentic)
- mutual agreement on a *session key* (i.e., a key that is valid only for one session) to prevent replay attacks
- An adversary that monitors the communication shall not be able to recover the session key.

Possible solution: next slide

B.46 (continued; simplified protocol fragment)



B.46 (continued)

Note: Neither the smart card nor the terminal can enforce a previous session key. The derivation of k_s requires the knowledge of k_C.

Both parties prove their authenticity in the remaining steps of the protocol by the application of k_s (implicit authentication).

B.47 Remark

- Usually encryption does not ensure data integrity.
- Data integrity is very important for many applications.
- Many applications require data integrity. Secrecy (privacy) is not mandatory if the exchanged (or stored) messages are not confidential.

B.48 MAC (Message Authentication Code)

Basic idea: Apply a block cipher with key k (to be kept secret) to generate a "control block" $MAC(p_1,p_2,\ldots,p_t,k)$ to the plaintext $p_1,p_2,\ldots,p_t$ with the following properties:

- Altering any plaintext bits changes the control block $MAC(p_1,p_2,\ldots,p_t,k)$ with overwhelming probability.
- The generation of the control block requires the knowledge of k. In particular, it shall not be feasible to construct valid MACs for new messages from known (plaintext, MAC) pairs.
- Checking the control block requires the knowledge of k.

B.49 Remark

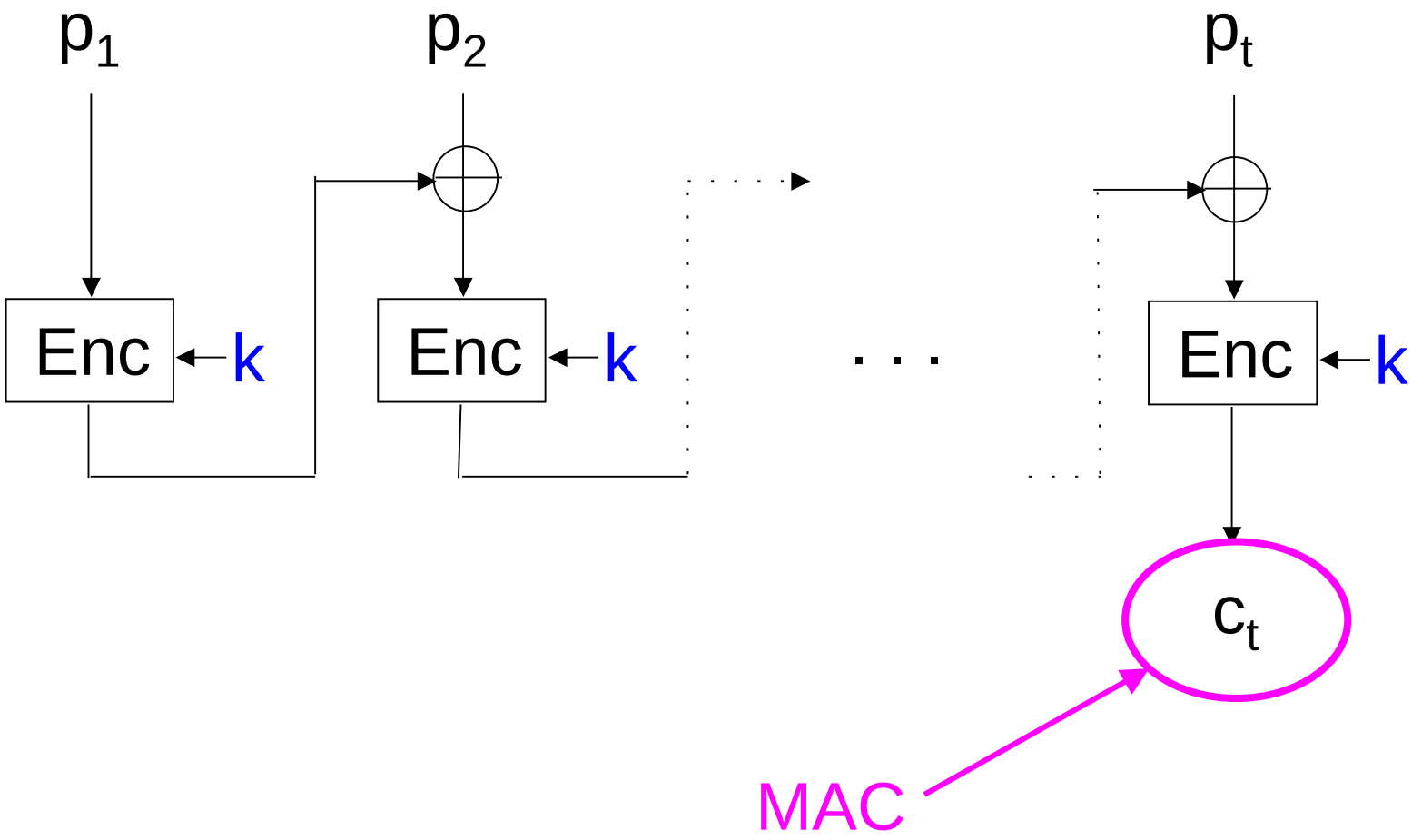
- If the “control block” MAC(p_1,p_2,…,p_t,k) does not “fit” to the plaintext p_1,p_2,…,p_t the receiver knows that the plaintext or the control block have been altered. The receiver will not accept this message.

Note: In Chapter C we will become acquainted with mechanisms that ensure data integrity where only the generation of the “control block” requires the knowledge of the secret key (digital signatures).

B.50 CBC - MAC

1st Step: Padding if necessary

2nd Step:



B.51 CBC - MAC: Security

- If Enc denotes a strong block cipher the CBC - MAC is secure for fixed-length plaintext messages (typical for smart card communication).
- The CBC - MAC is not secure if the block length of the exchanged messages is variable.

B.52 Attacking the CBC - MAC

Possible Attack: Assume that

- w m1:= MAC(p_1,p_2,…,p_t,k)
- w m2:= MAC(p_1*,p_2*,…,p_r*,k)

Then m2= MAC(p'_1,p'_2,…,p'_{t+r},k) with

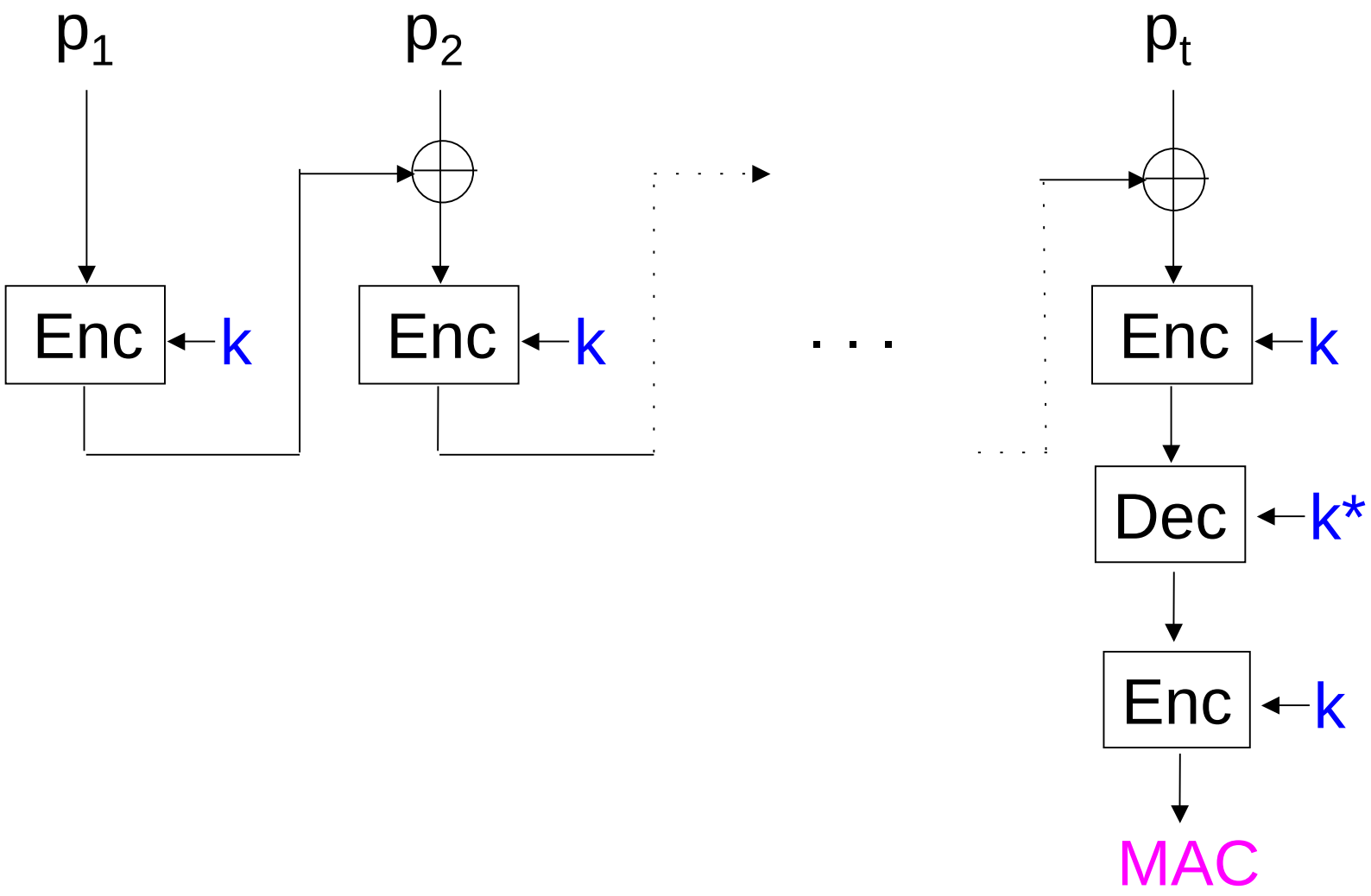
$p'_j = p_j$ for j=1,…,t

$p'_{t+1} = p_1^* \oplus m_1$

$p'_{t+j} = p_j^*$ for j=2,…,r

Details: Blackboard

B.53 strengthened CBC – MAC (Retail CBC - MAC)

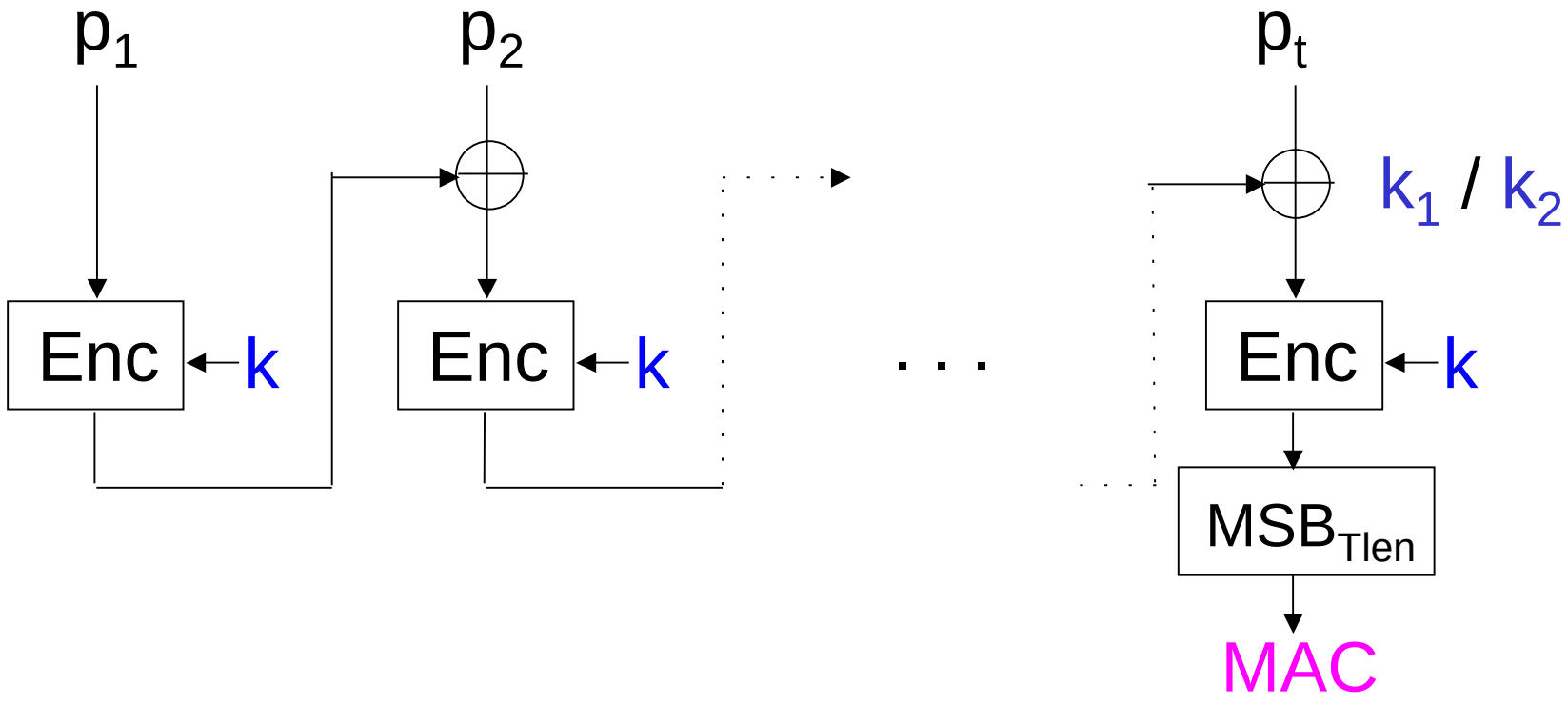


B.54 Retail CBC - MAC: Significant Properties

- The Retail CBC - MAC uses 2 keys.
- The Retail CBC - MAC is also secure for variable length plaintext messages (provided, of course, that Enc is a strong block cipher!).
- Note: Assume that $|K| \leq 2^n$. If the adversary knows about $2^{n/2}$ many (plaintext, MAC) pairs finding the keys k and k* is essentially only as difficult as finding the key k of an encryption with Enc.

B.55 CMAC

1st step: Padding if necessary: If the final block p_t is not complete it is filled up with the string "10…0".



The value k_1 is used if no padding has been necessary, otherwise k_2 is added. Moreover, Tlen ≤ n

B.56 Remark

- In 2005 the CMAC has been standardized by NIST.
- The CMAC is more efficient than the Retail-CBC-MAC as it saves one encryption and one decryption with Enc.
- The Retail CBC-MAC has been widespread in smart card applications for Enc = DES.

B.57 Remark

- To obtain MAC values that are shorter than n bits one may simply discard some of the right-most bits of the final ciphertext block (cf. B.55).
- In the literature a number of different MAC constructions have been studied.
- In Chapter C we will treat the HMAC. For efficiency reasons it is very important especially for long messages.

B.58 Example (cf. B.29)

- Electronic purse system: The smart card generates a payment record which is transmitted to the terminal. The merchant stores this record and submits it to the background system later.

B.58 (continued)

Security requirements: The payment record should

- be linked to the purse
- be linked to the terminal (making the theft of payment records useless)
- contain the transferred value (price)
- prevent the merchant from multiple submission of one payment record
- prevent replay attacks by dishonest customers
- only authentic purses shall be able to generate valid payment records
- the background system shall be able to detect any manipulations of payment records

B.58 (continued, simplified solution)

- Payment record (generated by the electronic purse)

$$\underbrace{C\_nr \,||\, T\_nr \,||\, value \,||\, trans\_nr \,||\, R}_{T} \,||\, MAC(T, k_{C(MAC)})$$

where

- C_nr: purse number
- T_nr: terminal number
- value: transferred value (price)
- trans_nr: transaction number ($\rightarrow$ terminal)
- R: random number (generated by the terminal)
- $k_{C(MAC)}$: card-individual key to compute MACs

B.58 (continued)

Exercise: Explain why this type of payment record meets the security requirements formulated on the previous slide.

B.59 Data integrity and secrecy

- MACs shall ensure data integrity.
- Encryption algorithms shall ensure secrecy.
- Both security mechanisms can be combined, e.g.:

$$(p_1,p_2,\ldots,p_t) \rightarrow \text{Enc}(p_1,p_2,\ldots,p_t,\text{MAC}(p_1,p_2,\ldots,p_t,k_1),k_2)$$
$$= c_1,c_2,\ldots,c_t,c_{t+1}$$

The receiver decrypts the message and checks whether the MAC is valid. A valid MAC convinces him that the message stems from the authentic sender and that the data have not been altered.

B.59 (continued)

For security reasons the sender should use different (more precisely: uncorrelated) keys for the MAC and the encryption.

B.60 Attack on Identical Keys

Assumptions:

- Encryption: CBC-MAC with IV=0 and key k_1
- CBC-MAC with k_2
- $k_1=k_2=k$

Observation:

- CBC-MAC($p_1,p_2,\ldots,p_t,k$) = Enc($c_{t-1} \oplus p_t,k$) = c_t (= t^{th} ciphertext block)

and

- Enc($p_1,p_2,\ldots,p_t$,CBC-MAC($p_1,p_2,\ldots,p_t,k$),k) =
  ($c_1,c_2,\ldots,c_t$,Enc(0,k))

B.60 (continued)

Identical keys for CBC-encryption (with IV = 0) and MAC computation reduce the overall computation time for both the sender and the receiver by about 50 %. In fact, since the CBC-MAC equals the ciphertext block c_t it need not be computed explicitly.

Unfortunately, an active adversary can mount an elementary attack (see next slide).

B.60 (continued)

Attack:

- The adversary alters or even drops some blocks c_j for $j < t-1$. Decryption yields the original CBC-MAC which equals the last but one block c_t of the encrypted message.
- The MAC clearly does not fit to the “new” plaintext.

However, with the control strategy described on the last slide the receiver will not detect the attack.

Details: Blackboard

B.60 (continued)

Exercise: Investigate whether this attack can be transferred into an attack against the Retail CBC-MAC if its first key k coincides with the encryption key (k^* is arbitrary).