Symmetric Key Encryption
Block Encryption Modes

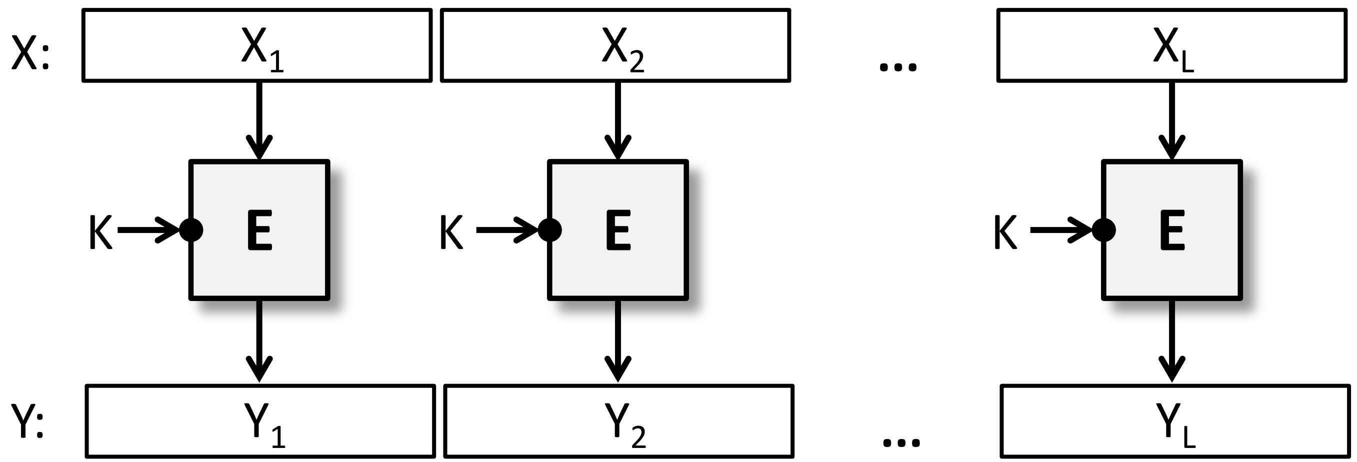
Levente Buttyán
CrySyS Lab, BME
buttyan@crysys.hu

Block cipher usage modes

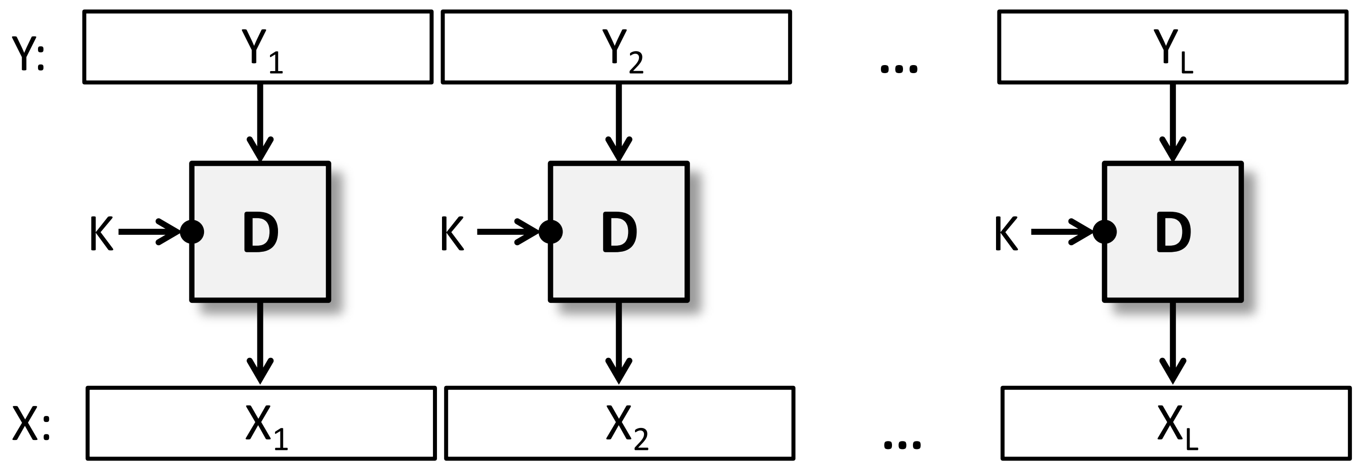
- we usually need to encrypt messages that are longer (or shorter) than the block size of the block cipher
- we can use the block cipher in different ”encryption modes”
- basic modes
  - **Electronic Codebook (ECB) mode**
  - **Cipher Block Chaining (CBC) mode**
  - Cipher Feedback (CFB) mode
  - Output Feedback (OFB) mode
  - **Counter (CTR) mode**
- authenticated encryption modes (will be covered later)
  - CCM: CTR + CBC MAC
  - GCM: Galois CTR mode
  - OCB: Offset Codebook Mode

ECB mode

- encrypt:



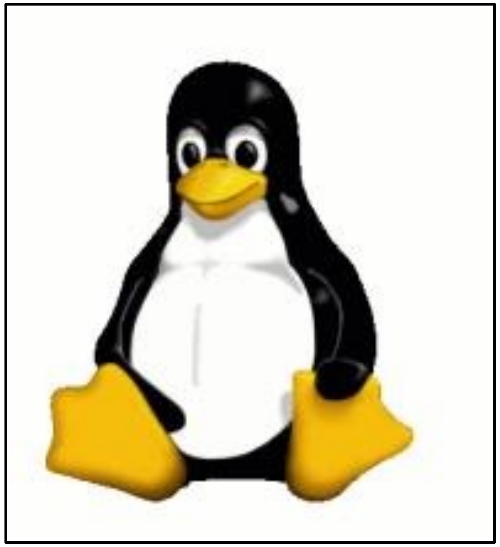
- decrypt:



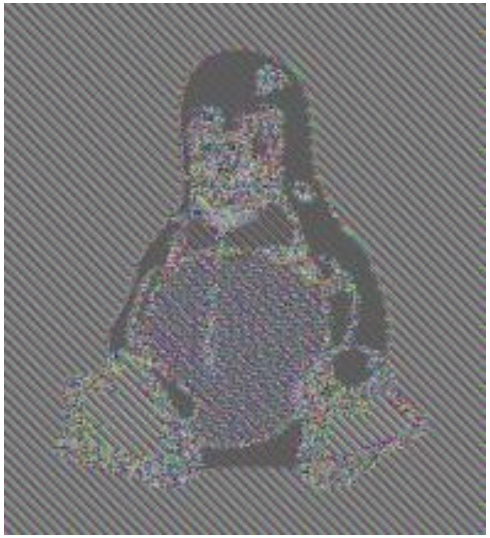
Properties of ECB mode

- encrypting the same plaintext with the same key results in the same ciphertext (no randomization)
- identical plaintext blocks result in identical ciphertext blocks (under the same key of course)
  - messages to be encrypted often have very regular formats
  - repeating fragments, string of 0s, ... are quite common
- **does not properly hide patterns in the plaintext**
- overall: not recommended for messages longer than one block, or if keys are reused for more than one block

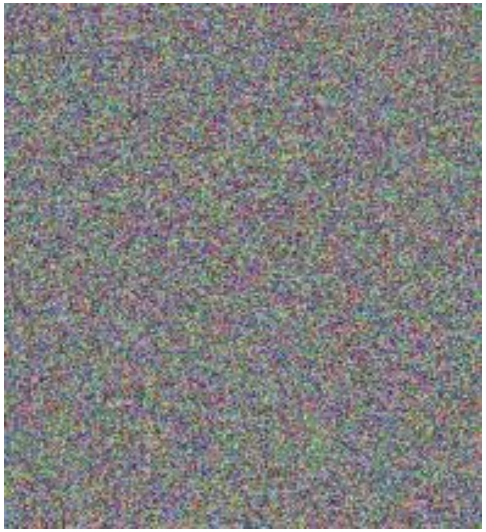
Illustration of ECB's weakness



original image

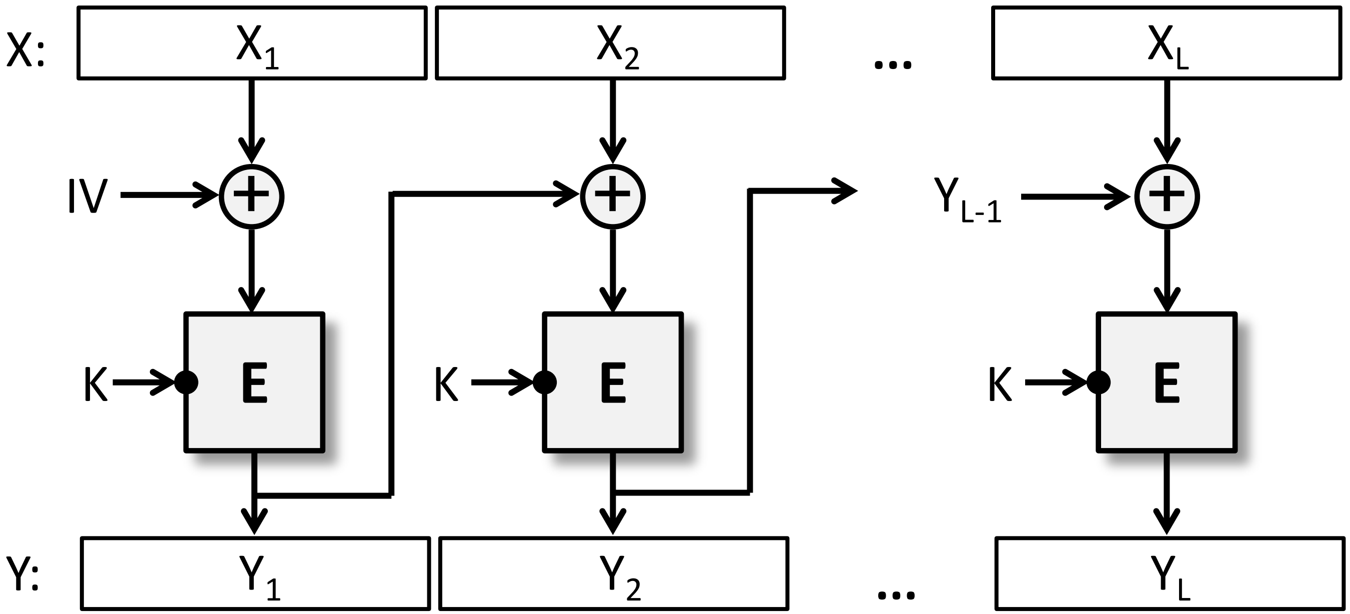


encrypted in ECB mode



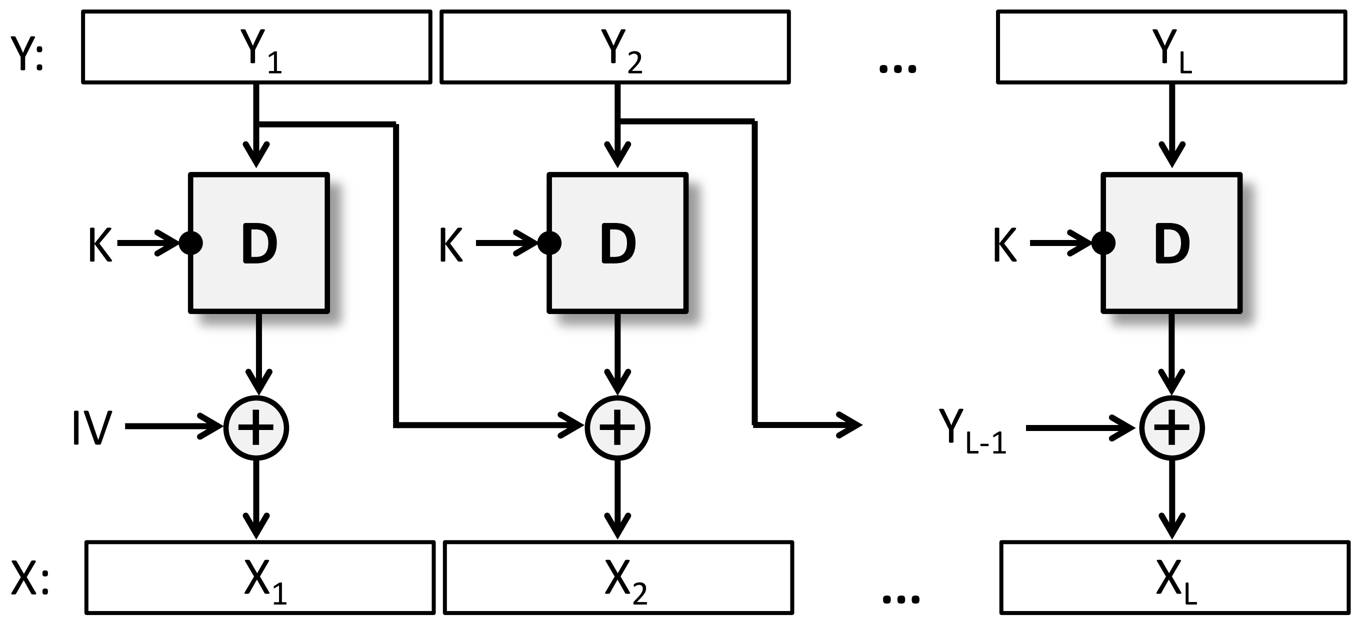
using other modes

CBC mode (encryption)



$$Y_i = E_K(X_i \oplus Y_{i-1})$$

CBC mode (decryption)



$$\boxed{X_i = D_K(Y_i) \oplus Y_{i-1}}$$

Properties of CBC mode

- encrypting the same plaintexts under the same key, but different IVs result in different ciphertexts
- ciphertext block Y_i depends on X_i and all preceding plaintext blocks
  - however, dependency on the preceding plaintext blocks is only via the previous ciphertext block Y_{i-1}
  - hence, proper decryption of a correct ciphertext block needs a correct preceding ciphertext block only (**chosen ciphertext attacks are possible**)
- self-synchronizing property:
  - automatically recovers from loss of a ciphertext block
- parallel computation (only for decryption), random access, no pre-computation

Requirements on the IV

- the IV size is the block size of the cipher
- the IV needs to be transferred to the receiver
  - it is usually sent at the beginning of the message
- the IV does not need to be secret, but it **must be unpredictable** by the attacker
  - the problem with predictable IVs will be explained later
- it is also advantageous if the IV cannot be manipulated at will...

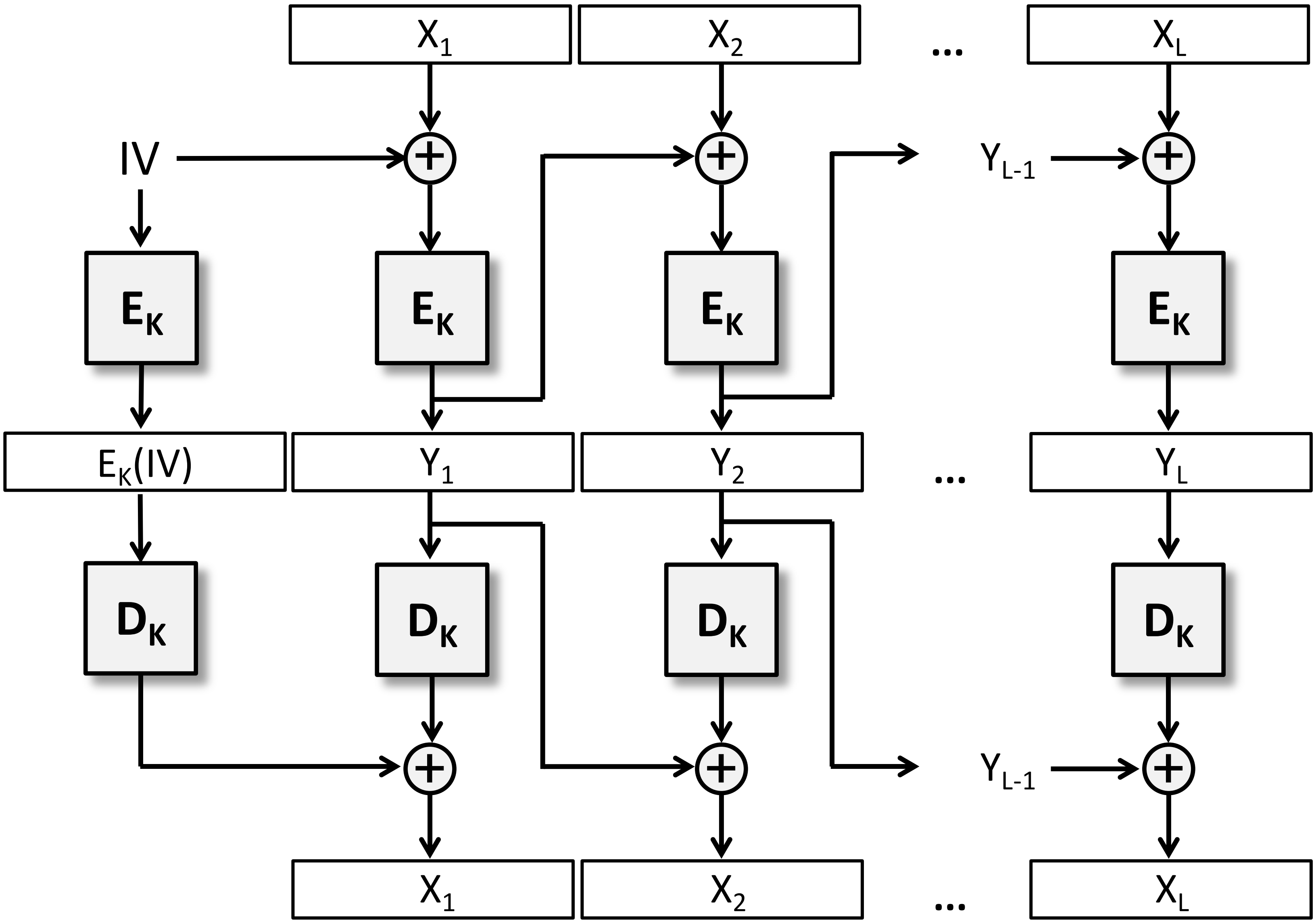
Generating unpredictable IVs

- IV = output of a **cryptographic** random number generator
  - random number generators available in standard programming libraries (e.g., rnd, rand, …) are not unpredictable, therefore they are not appropriate here!
  - cryptographic random number generators are unpredictable
  - to also ensure non-manipulability, the sender should send the IV in an encrypted form (e.g., $E_K(IV)$) to the receiver

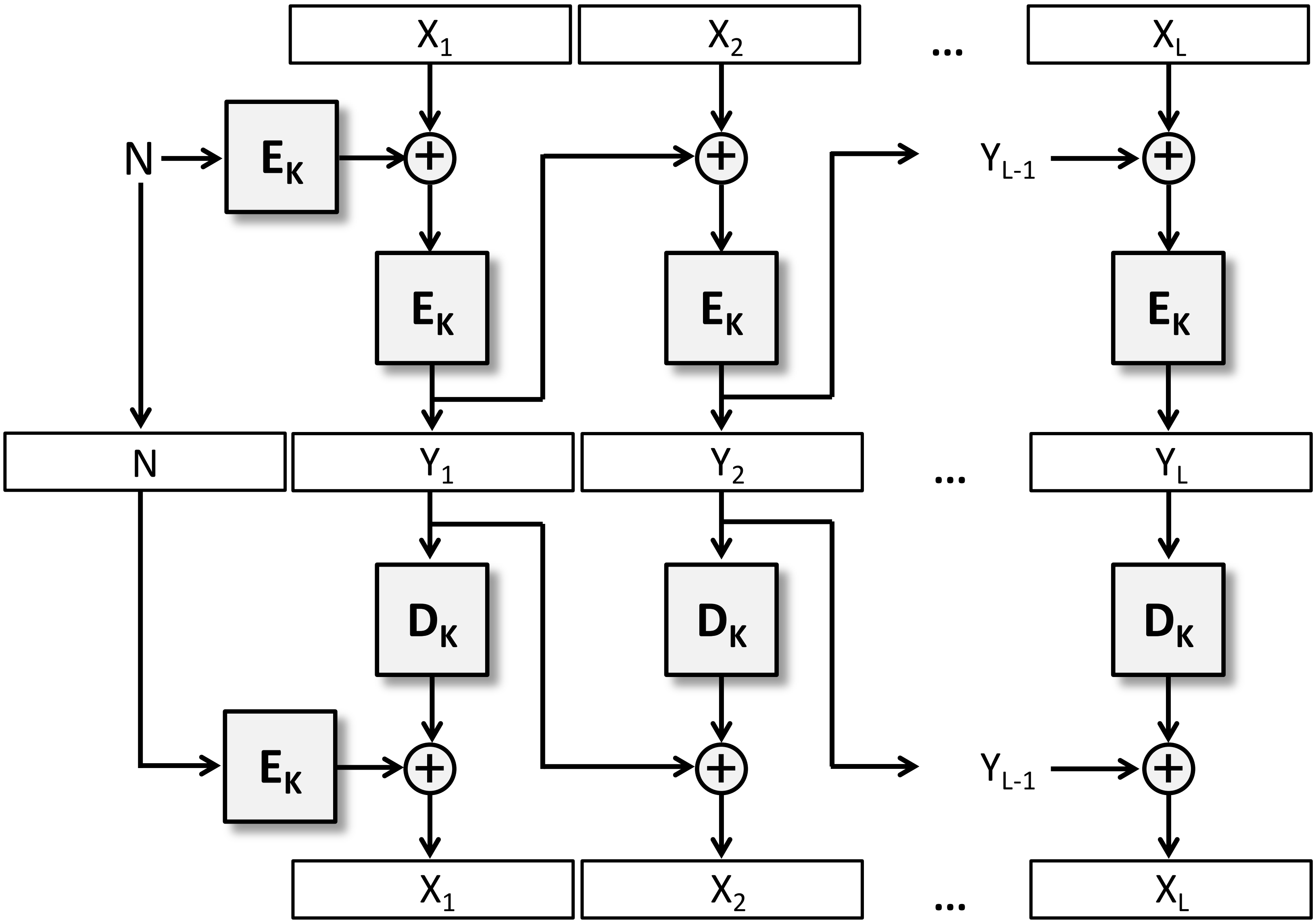
- IV = $E_K(N)$
  - where N is a nonce (”number used once”)
  - N may be a counter or a message ID (unique across messages), which may be sent by the sender to the receiver (perhaps at the beginning of the CBC encrypted message)
  - to ensure unpredictability and non-manipulability, the receiver should then compute the IV locally as IV = $E_K(N)$

- as a side effect, both approaches also ensure the secrecy of IV

IV as a random number – illustrated



IV as an encrypted nonce – illustrated

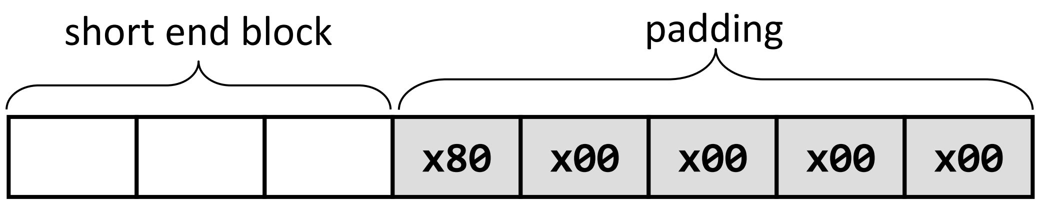


Padding

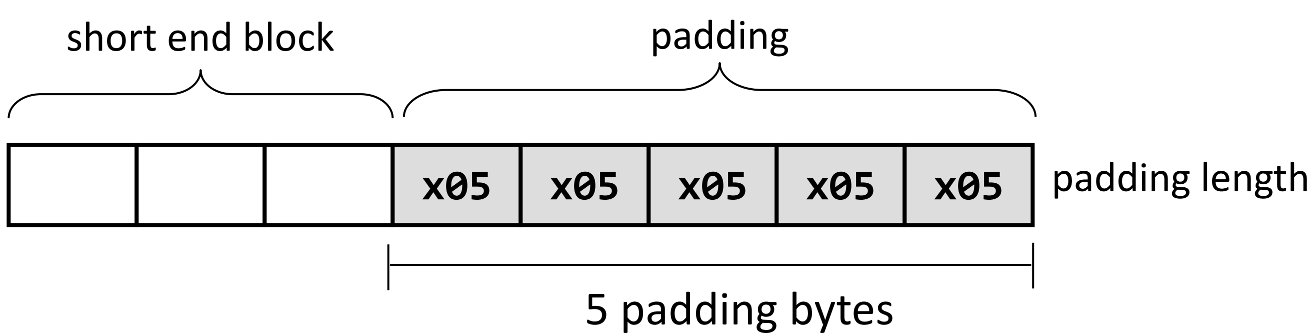
- the length of the message may not be a multiple of the cipher’s block size
- we must add some extra bytes to the short end block such that it reaches the correct size – this is called **padding**
- the receiver must be able to unambiguously recognize and remove the padding
- due to this unambiguity requirement, padding is actually always used, even in the case when the length of the original message is a multiple of the block size (in this case, an entire extra block is added to the message)

Common padding schemes

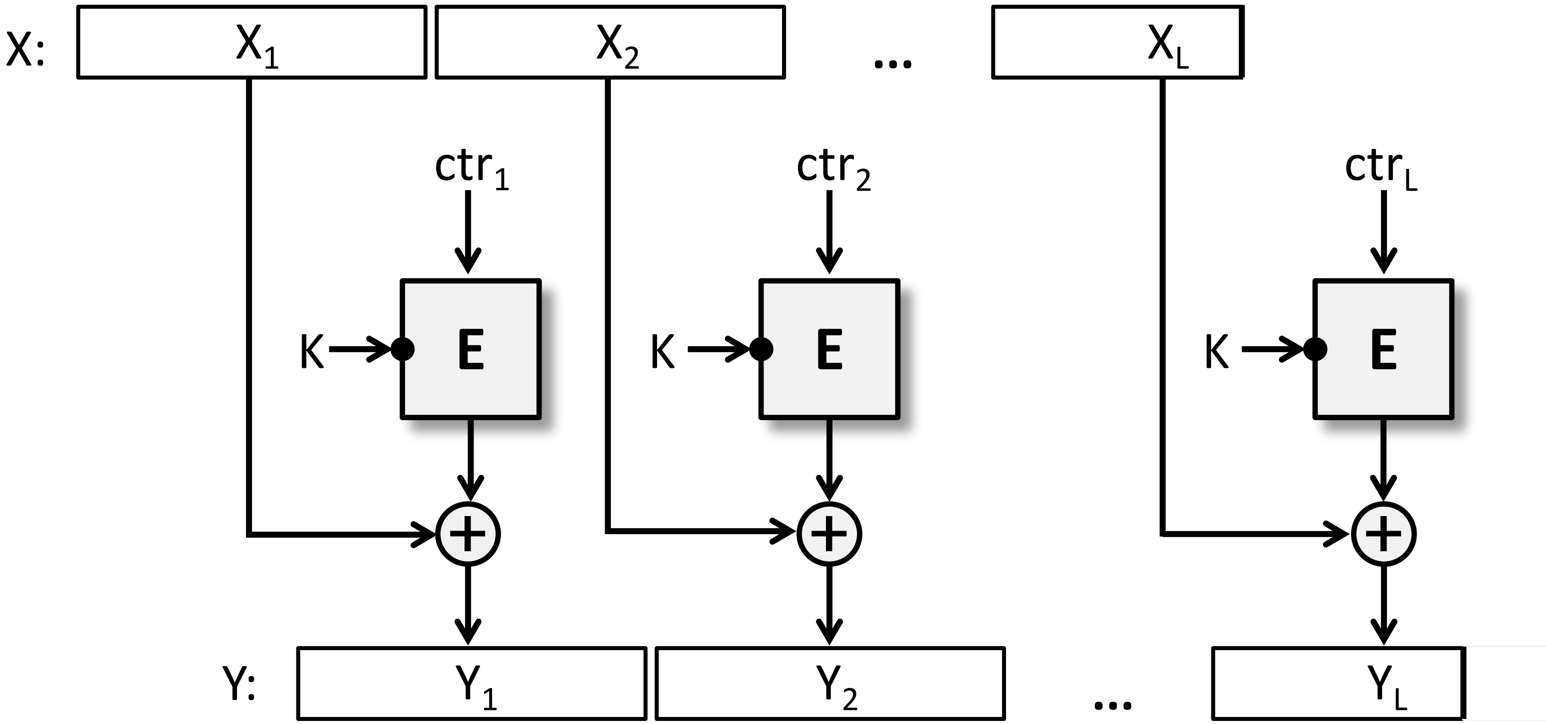
- append a x80 byte and then as many x00 bytes as needed [ISO 7816-4]



- indicate the length of the padding in the last added byte [ANSI X.923, PKCS#7]
  - padding bytes can be random (SSL padding)
  - padding bytes can be requiered to have special values
    - » e.g., padding length byte repeated (TLS padding)

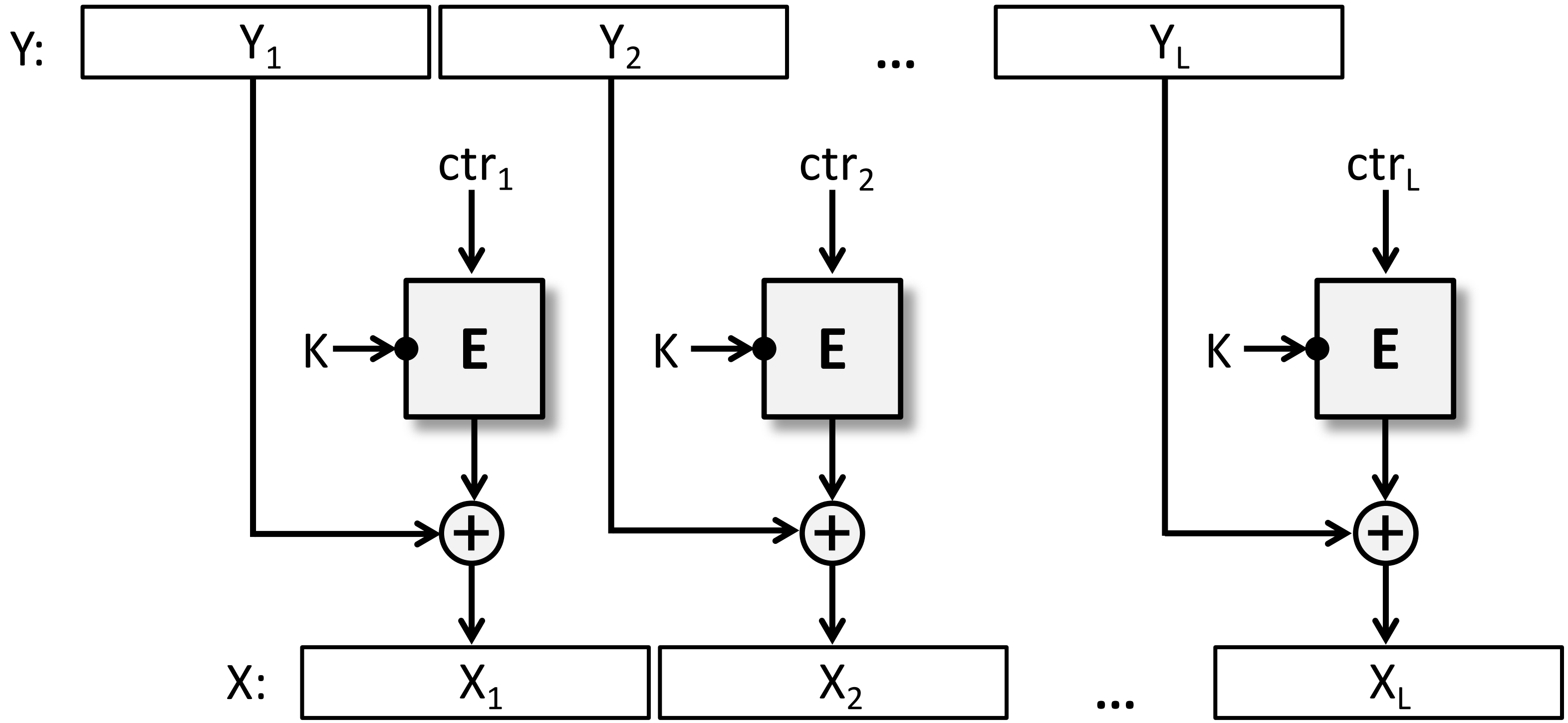


CTR mode (encryption)



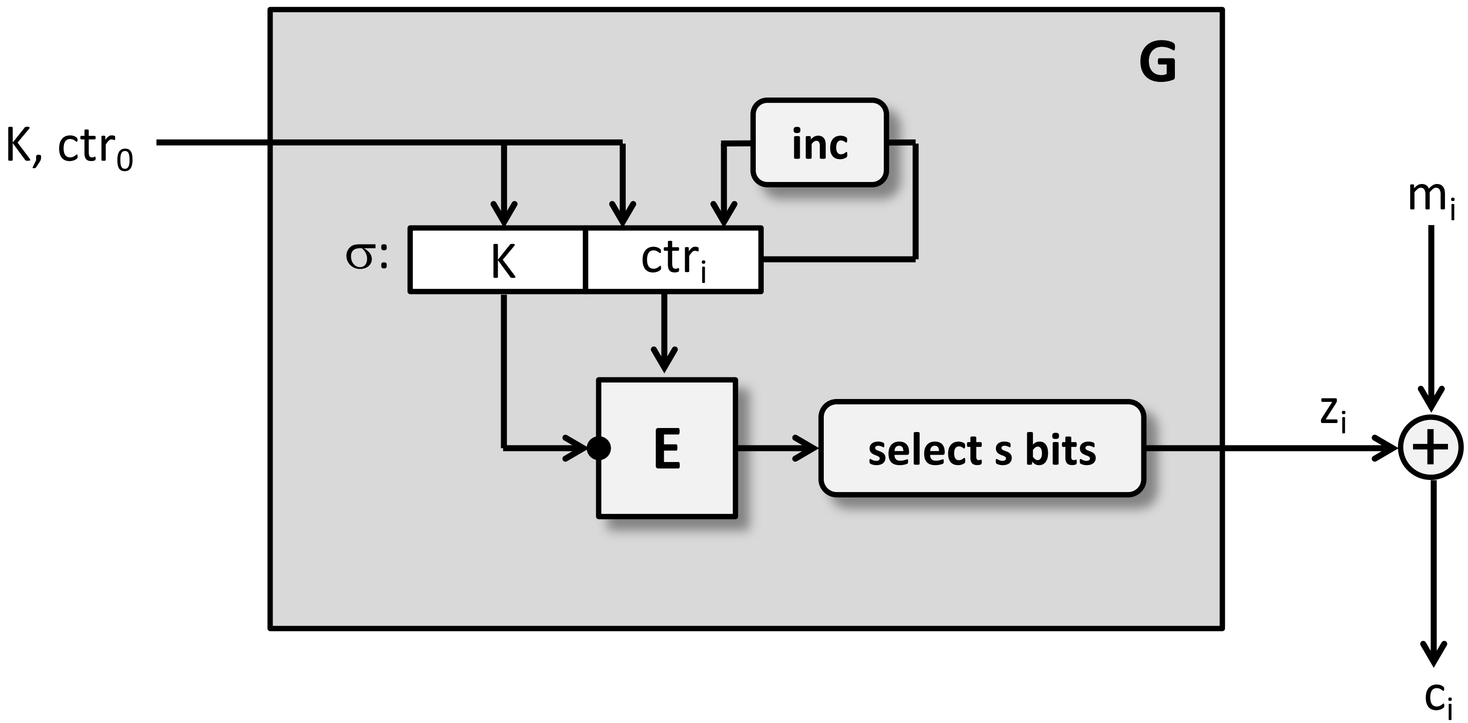
$$Y_i = X_i \oplus E_K(ctr_i)$$
$$ctr_{i+1} = ctr_i + 1$$

CTR mode (decryption)



$$X_i = Y_i \oplus E_K(ctr_i)$$
$$ctr_{i+1} = ctr_i + 1$$

Another view on CTR



this is a stream cipher where

- the internal state is (K, ctr_i)
- the update function increments the counter
- the generator function uses the block cipher

Properties of CTR mode

- it is crucial that counter values do not repeat, otherwise…
  - given $Y = E_K(ctr) + X$ and $Y' = E_K(ctr) + X'$, the attacker can compute $Y + Y' = X + X'$
  - hence, if X (or part of it) is known then X' (or part of it) is disclosed
- does not provide any integrity protection
  - similar to the case of steam ciphers, an attacker can perform controlled modifications on the plaintext by manipulating the ciphertext
- does not increase message length (unlike CBC with padding)
- in addition, CTR mode is parallelizable, and allows for random access and pre-computation

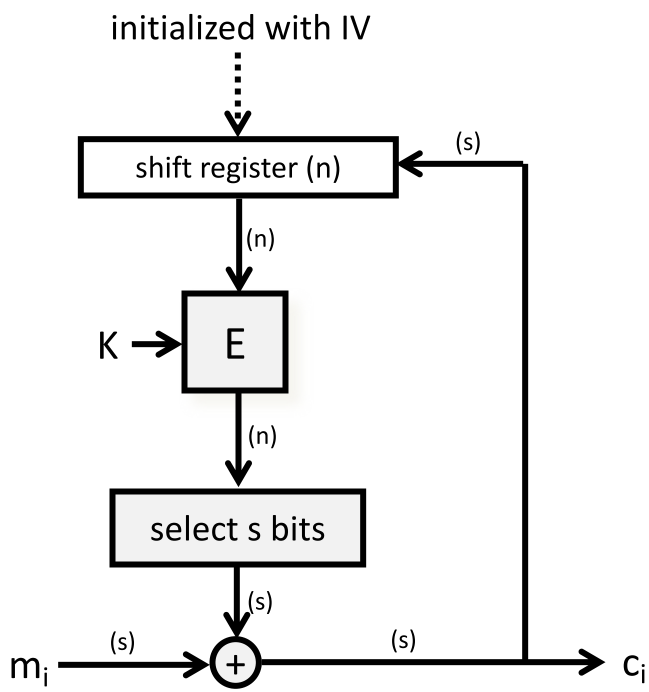
Generating non-repeating counters

- requirements:
  - counter values should not repeat within a given message
  - initial counter must be chosen to ensure that counters are unique across all messages that are encrypted under the given key

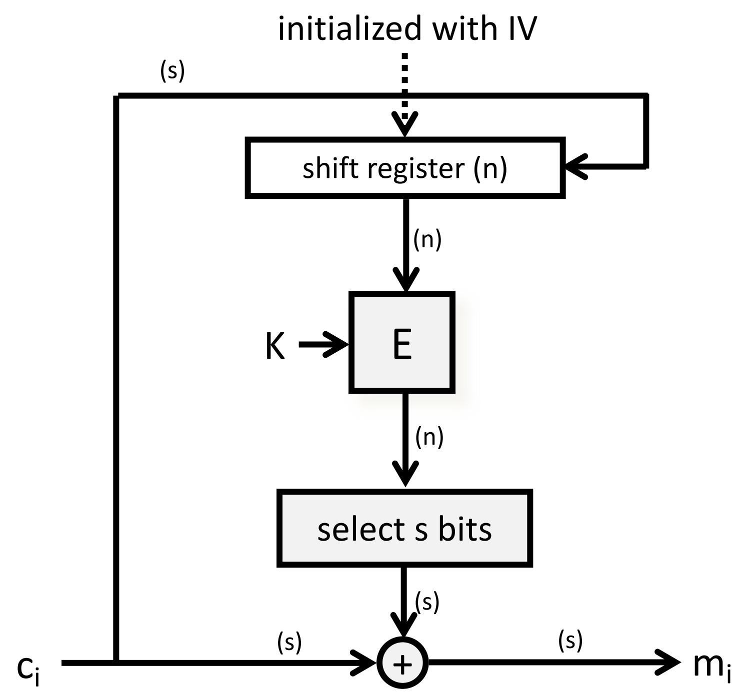
- a typical approach:
  - divide the counter block into two sub-blocks ctr = ctr’|ctr’’, where ctr’’ is b bits long and ctr’ is n-b bits long (n is the block size of the cipher)
  - ctr’ is a unique message ID or message counter incremented with each new message (→ max number of messages is 2^{n-b})
  - ctr’’ is a counter incremented with every block within the message (→ max message length is 2^b blocks)

CFB mode

– encrypt

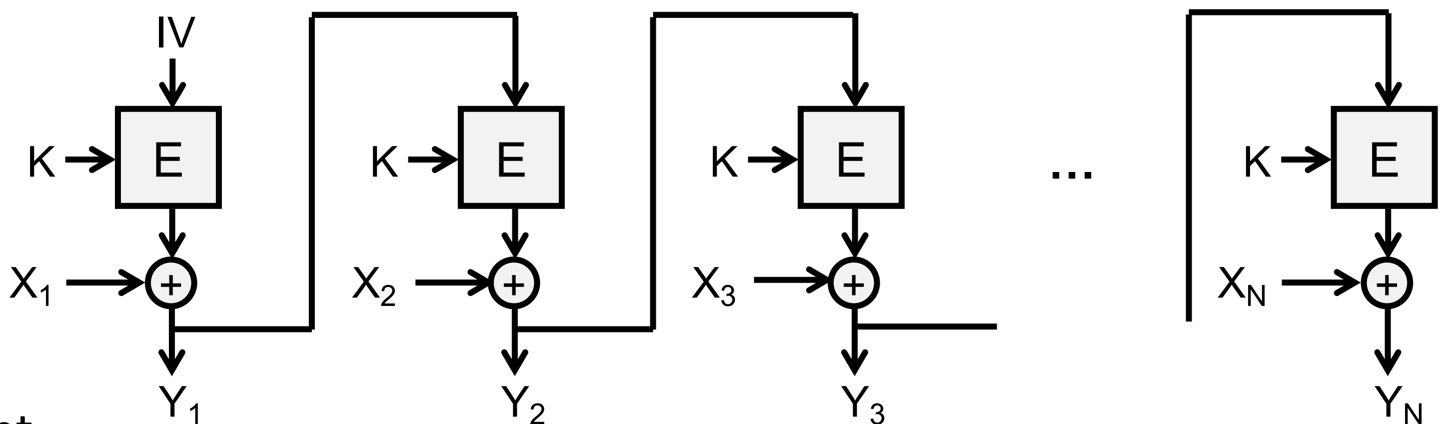


– decrypt

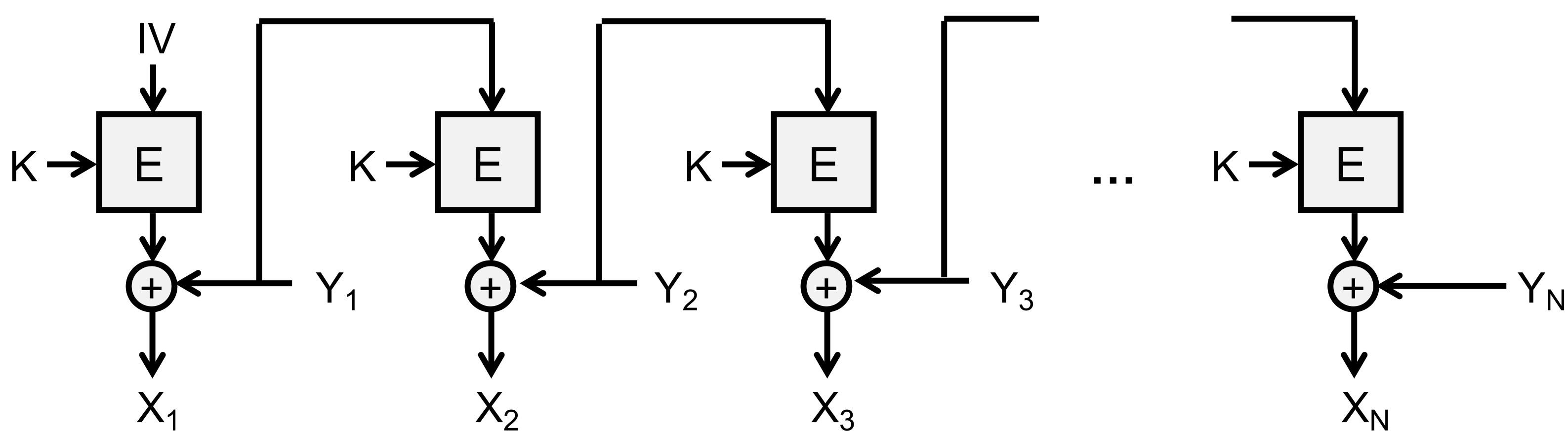


Another view on CFB

- if s = n, then…
  - encrypt



  - decrypt



Properties of CFB mode

- encrypting the same plaintexts under the same key, but different IVs results in different ciphertexts

- the IV can be sent in clear, and need not be unpredictable

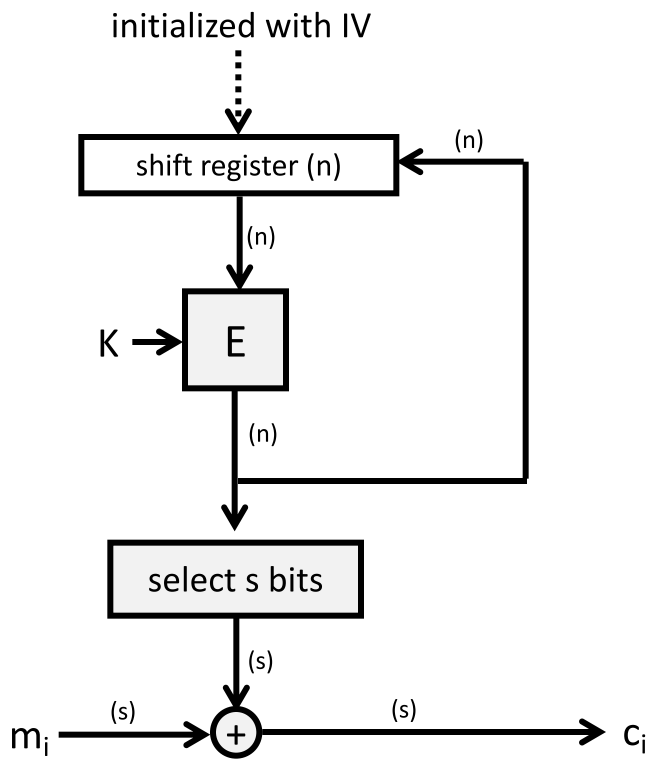
- ciphertext character c_j depends on m_j and all preceding plaintext characters
  - however, proper decryption of a ciphertext character needs only the preceding n/s ciphertext characters to be correct

- self-synchronizing property:
  - recovers from loss of a ciphertext character after n/s steps

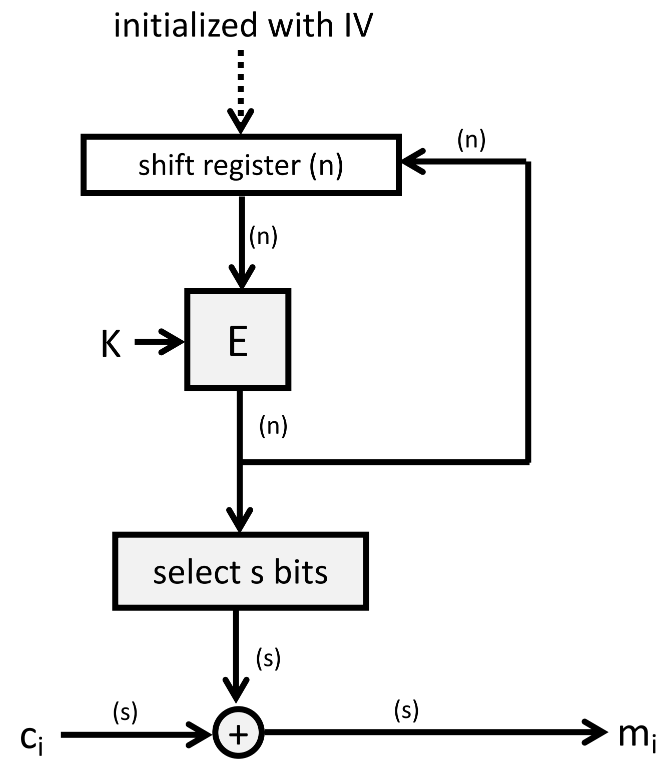
- parallel computation (only for decryption), random access, no pre-computation

OFB mode

– encrypt

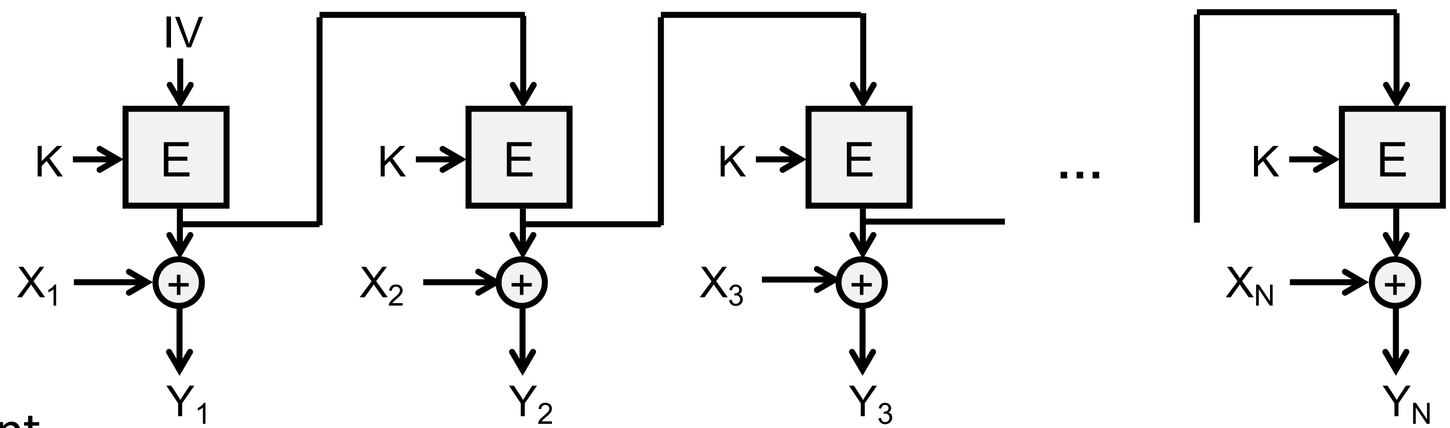


– decrypt

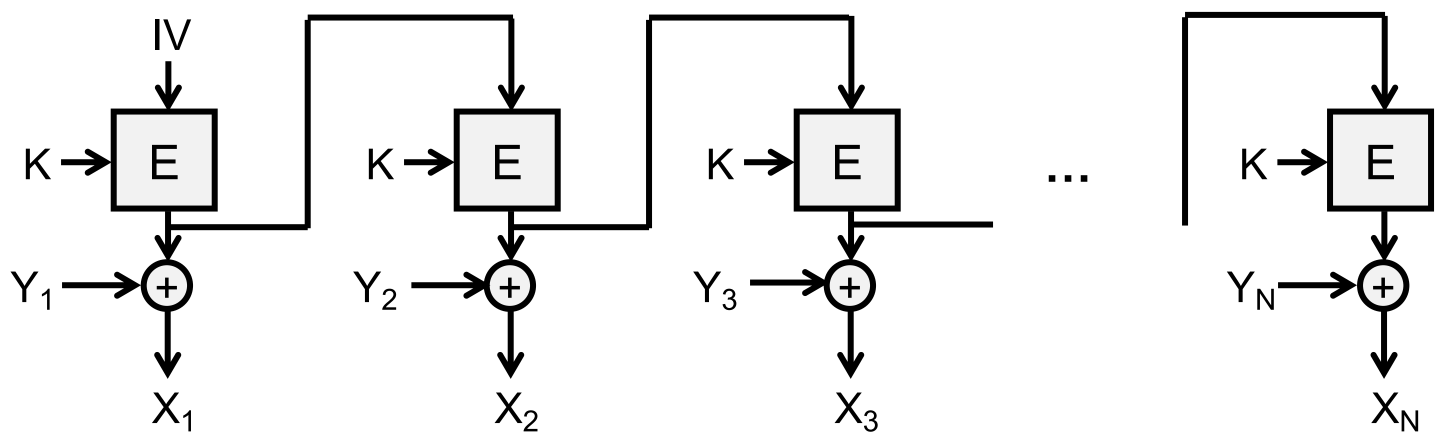


Another view on OFB

- if s = n, then…
  - encrypt



  - decrypt



Properties of OFB mode

- a different IV should be used for every new message, otherwise messages will be encrypted with the same key stream!
- the IV can be sent in clear
  - however, if the IV is modified by the attacker, then the cipher will never recover (unlike CFB)
- needs synchronization
  - cannot automatically recover from a loss of a ciphertext character
- sequential computation only, no random access, pre-computation is possible
- no integrity at all: an attacker may cause controlled bit changes in any plaintext character!

Summary on basic modes

- ECB: used to encipher a single plaintext block
  - e.g., an AES key or an IV
- CBC: repeated use of the block cipher to encrypt long messages
  - IV should be changed for every message
  - the unpredictability of the IV is important
  - only the decryption can be parallelized, random access, no pre-computation
  - self-synchronizing property
- CTR, CFB, OFB:
  - can be used to convert a block cipher into a stream cipher ($s < n$)
    - » CTR and OFB: synchronous stream ciphers
    - » CFB: self-synchronizing stream-cipher
  - only the encryption algorithm is used, that is why some block ciphers (e.g., AES) are optimized for encryption

Summary on basic modes

- CTR:
  - non-repeating counters are very important
  - parallelizable, random access, pre-computation
  - needs synchronization

- CFB:
  - IV should be changed for every message
  - only the decryption can be parallelized, random access, no pre-computation
  - self-synchronizing property

- OFB:
  - changing the IV for every message is very important
  - cannot be parallelized, no random access, pre-computation is possible
  - needs synchronization

- **none of these modes provide integrity protection !**
- in CBC mode, encrypted message is longer than clear message due to padding

Control questions

- Why do we need block encryption modes?
- What is the main weakness of the ECB mode?
- How do the CBC and the CTR modes work and what are their main properties?
  - strength and weaknesses?
  - issues with IVs (CBC) and counters (CTR)?
  - padding?
  - self-synchronization?
  - random access, parallel computation, pre-computation?
- How to generate unpredictable IVs for CBC?
- How to generate non-repeating counters for CTR?