

## Modes of Operation

How to encrypt large messages  
using block ciphers

---

---

---

---

---

---

---

---

## Large messages

- In order for encryption to be cost-efficient, a key  $K$  should be used to encrypt data of total size much larger than  $K$ .
  - Typical key sizes are in the range of 80-512 bits while blocks are often 64-128 bits.
- Multiple blocks should be encrypted under the same key.
  - How to do this efficiently and securely?

---

---

---

---

---

---

---

---

## Dividing a message into chunks

- Given a message  $m$ , and blocksize  $b$ , there should be at least  $\lceil m/b \rceil$  blocks to encrypt. The last block may not be full. It will have to be completed with **padding** bits.
  - The padding needs to be uniquely detectable and reversible.
  - Example that does NOT work: Fill the rest of the block with 0's.

---

---

---

---

---

---

---

---

## Padding: PKCS5

- Let  $n$  be the length of the data after the last full data block.
- $0 \leq n \leq b-1$ , where  $b$  is blocksize in bits/bytes.
- Fill the rest of the block with repeated copies of  $(b-n)_2$ , the binary representation of the number of padding bits/bytes.
- If  $n = 0$ , i.e., if the message is an exact multiple of the blocksize, attach a whole new block of  $(b)_2$ 's to the end of the message.

---

---

---

---

---

---

---

---

## Padding data for DES

- DES has 64-bit/8-byte blocksize.
- If the message data is a bytestream, take for  $n$  the number of **bytes** of the length of data after last full data block.  $n = 0, 1, \dots, 7$ , only.
- Use a full byte to encode  $(8 - n)_2$ :
  - If  $n = 3$ ,  $(8-n)_2 = (00000101)_2$ . The last block has first 3 bytes equal to the last three data bytes, followed by 5 repetitions of  $(00000101)_2$
  - If  $n = 0$ ,  $(8-n)_2 = (00001000)_2$ , and the last block has 8 copies of this.

---

---

---

---

---

---

---

---

## Removing the padding

- For DES-PKCS5, read last byte  $B$ .
- If  $B$  does not represent an integer in  $\{1, 2, \dots, 8\}$ , report FAILURE.
- Discard the last  $B$  bytes of the last block.
- Questions for thinking:
  - Why for this padding is it necessary to add a whole padding block to messages that have length an exact multiple of the blocksize?
  - Is it possible to design a uniquely decodable padding that does not need to pad an exact-fitting message?

---

---

---

---

---

---

---

---

## ECB

- The simplest way to encrypt data using a block cipher is to encrypt each data block separately: **Electronic Code Book (ECB) mode**.
- Not secure for large messages:
  - If plaintext blocks ever repeat, their corresponding ciphertext blocks are equal.
  - facilitates given- and chosen-plaintext/ ciphertext attacks.
  - reasonable for small amounts of random data, such as an initialization vector (used by other data encryption modes), other keys, etc.

---

---

---

---

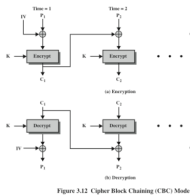
---

---

---

---

## CBC



- **Cipher Block Chaining mode**
- Requires an IV (initialization vector)
- $c_0: IV; c_{i+1} = E(p_{i+1} \oplus c_i)$
- $p_{i+1} = D(c_{i+1}) \oplus c_i$
- $c_i$ : encryption of the  $i$ -th plaintext block  $p_i$
- $E(), D()$ : basic (ECB) encryption, decryption

---

---

---

---

---

---

---

---

## CBC properties

$$p_{i+1} = D(c_{i+1}) \oplus c_i$$

- Secure for large messages within limits.
- ECB is self-synchronizing:
  - If block  $c_i$  is lost during transmission, the following block will not decrypt correctly ( $c_i$  is needed to decrypt  $c_{i+1}$ ). However,  $c_{i+2}$  will decrypt correctly (if  $c_{i+1}$  is received).
- Error propagation rate ( $1 \rightarrow b+1$ ):
  - If  $j$ -th bit of  $c_i$  is received incorrectly, the whole of  $p_i$  decrypts incorrectly, as well as the  $j$ -th bit of  $p_{i+1}$ . Later blocks are not affected.

---

---

---

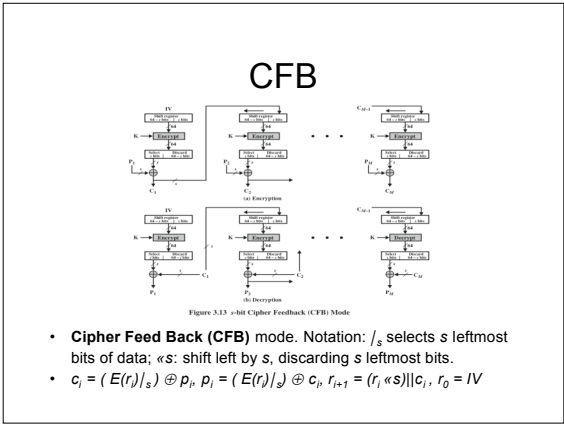
---

---

---

---

---




---

---

---

---

---

---

---

---

### CFB properties

$$p_{i+1} = (E(r_i) \lfloor_s) \oplus c_i$$

$$r_{i+1} = (r_i \ll s) \parallel c_i$$

- Secure for large messages within limits.
- CFB decryption re-uses ECB encryption for decryption: Shorter code.
- CFB error-propagation ( $1 \rightarrow 1 + R$ ):
  - If  $j$ -th bit of block  $c_j$  (counting from right) is received incorrectly, the  $j$ -th bit of  $p_j$  is corrupted. A further  $s \cdot k$  bits will be corrupted, where  $k$  is the smallest number such that  $j + s \cdot k \geq R$ , the register buffer size. In the worst case  $j = 0$ , and  $s \cdot k = R$ . (Assuming  $R$  is a multiple of  $s$ , a common case.)

---

---

---

---

---

---

---

---

### More CFB properties

- The value  $s$  can be tuned to eliminate the need for padding.
  - For instance,  $s = 8$  for a data bytestream.
  - Smaller amounts of data can be independently encrypted ( $s=8$  allows for encryption of single bytes):
    - Adequate method for streaming data (no need to buffer data until blocklength bits of data are available for transmission. *However, that implies an added cost: one ECB encryption "per  $s$  bits" instead of "per block."*)
- CFB is self-synchronizing.

---

---

---

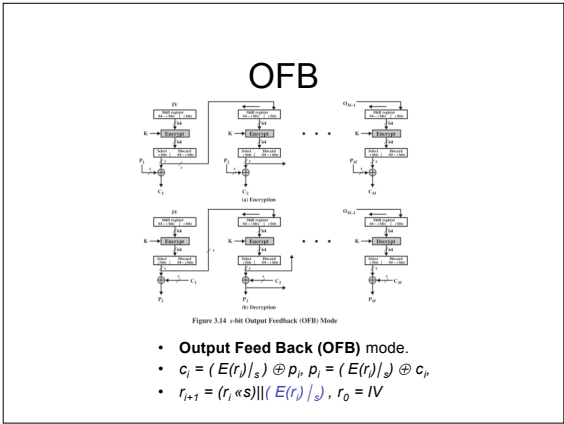
---

---

---

---

---




---

---

---

---

---

---

---

---

---

---

### OFB properties

$$p_{i+1} = (E(r)_s) \oplus c_i$$

$$r_{i+1} = (r_i \ll s) \parallel (E(r)_s)$$

- Secure for large messages within limits
- OFB decryption re-uses ECB encryption for decryption: Shorter code.
- OFB error-propagation (1 → 1):
  - OFB does not feed the received ciphertext in its register so a wrong ciphertext bit only affects the same bit of the plaintext.
- OFB does NOT self-synchronize. Instead it requires synchronization: it is a **synchronous** mode.
  - The register at the receiver will be permanently ahead of the register at the sender if a ciphertext block is lost in transmission.

---

---

---

---

---

---

---

---

---

---

### More OFB properties

- As in CFB, OFB can be used with streaming data, by tuning the value s. Also, if s is as large as the smallest data unit, it does not require padding.

---

---

---

---

---

---

---

---

---

---

## Counter mode

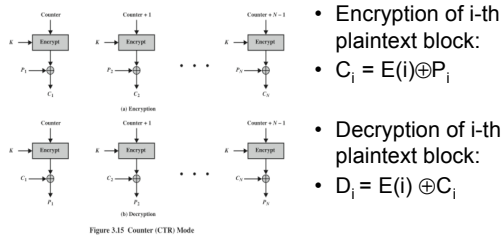


Figure 3.15 Counter (CTR) Mode

- Encryption of  $i$ -th plaintext block:
- $C_i = E(i) \oplus P_i$

- Decryption of  $i$ -th plaintext block:
- $D_i = E(i) \oplus C_i$

---

---

---

---

---

---

---

---

## Counter mode properties

- Error propagation (1→ 1): Each corrupted bit of ciphertext results in the same bit corrupted after decryption
- Synchronous (requires synchronization)
- Enables pre-computation of keystream
- Fully parallelizable
- Random access mode
- Security as good as other modes (!)

---

---

---

---

---

---

---

---

## Notes of caution

- An pair (key, IV) should never be used to encrypt more than one message
- Some modes (like counter) require only that IV be not re-usable. Other modes require that IV be unpredictable (CBC)
- Encryption does not provide integrity protection. This is particularly problematic with OFB/counter. Why?

---

---

---

---

---

---

---

---