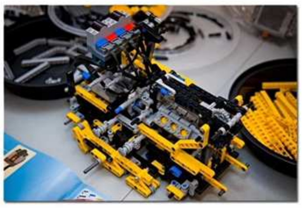
Block cipher modes

Security Protocols (bmevihim132)

Dr. Levente Buttyán
associate professor
BME Hálózati Rendszerek és Szolgáltatások Tanszék
Lab of Cryptography and System Security (CrySyS)
buttyan@hit.bme.hu, buttyan@crysys.hu

Outline

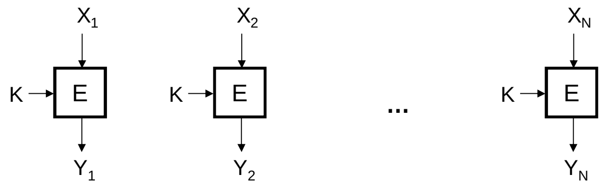
- five standardized modes (operation, properties)
  - Electronic Codebook (ECB) mode
  - Cipher Block Chaining (CBC) mode
  - Cipher Feedback (CFB) mode
  - Output Feedback (OFB) mode
  - Counter (CTR) mode



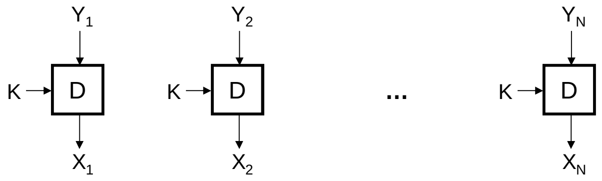
- attacks on CBC
  - simple attacks (content leak, cut and paste)
  - padding oracle attack by Vaudenay (2002)
- an attack on the CFB variant used in OpenPGP
- some exercises

ECB mode

- encrypt



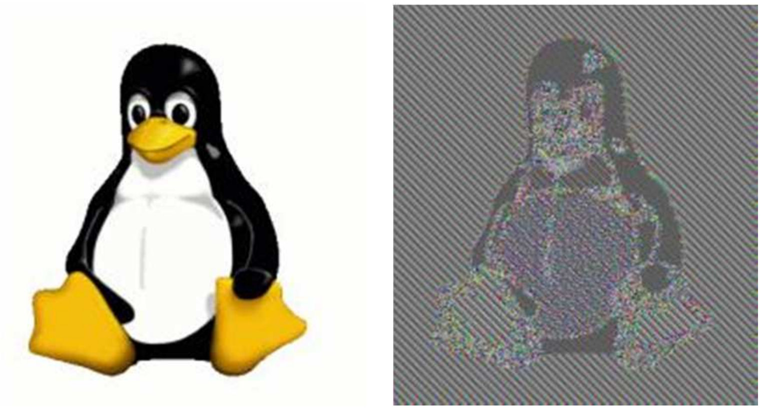
- decrypt



Properties of the ECB mode

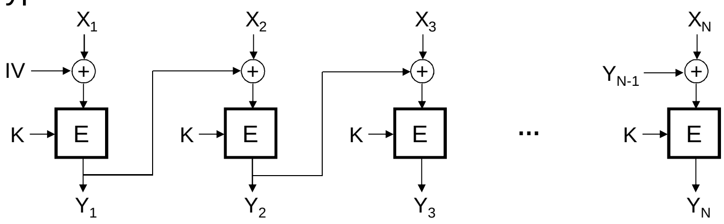
- encrypting the same plaintext with the same key results in the same ciphertext
- identical plaintext blocks result in identical ciphertext blocks (under the same key of course)
  - messages to be encrypted often have very regular formats
  - repeating fragments, special headers, string of 0s, etc. are quite common
  - **does not properly hide patterns in the plaintext**
- blocks are encrypted independently of other blocks
  - reordering ciphertext blocks result in correspondingly reordered plaintext blocks
  - ciphertext blocks can be cut from one message and pasted in another, possibly without detection
  - **additional integrity protection is essential**
- error propagation: one bit error in a ciphertext block affects only the corresponding plaintext block (results in garbage)
- overall: not recommended for messages longer than one block, or if keys are reused for more than one block

Illustration of ECB in action

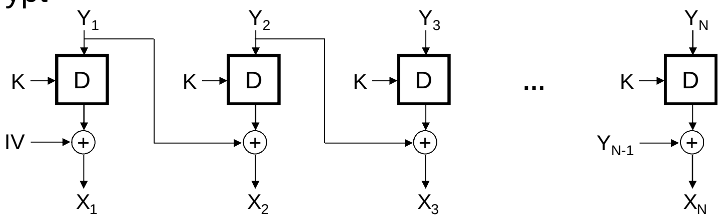


CBC mode

- encrypt



- decrypt



Properties of the CBC mode

- encrypting the same plaintext under the same key, but different IVs result in different ciphertexts
- ciphertext block Y_j depends on X_j and all preceding plaintext blocks
  - rearranging ciphertext blocks affects decryption
  - however, dependency on the preceding plaintext blocks is only via the previous ciphertext block Y_{j-1}

  → proper decryption of a correct ciphertext block needs a correct preceding ciphertext block only (see cut-and-paste attacks later in this slide set)
- error propagation:
  - one bit error in a ciphertext block Y_j has an effect on the j-th and (j+1)-st plaintext block
    - X_j' is complete garbage and X_{j+1}' has bit errors where Y_j had

    → an attacker may cause predictable bit changes in the (j+1)-st plaintext block (see the padding oracle attack later in this slide set)
- 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

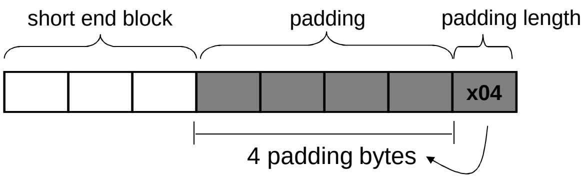
- the IV need not be secret (although secret IVs have some advantages), but it should be **unpredictable** and **non-manipulable** by the attacker
- the problem with predictable IVs (in the chosen plaintext attack model)
  - let $Y_i = E_K(Y_{i-1} + X_i)$ for some i (part of a CBC encrypted message), and let us assume that the attacker suspects that $X_i = X^*$; can he confirm this?
  - the attacker predicts the next IV, submits $X = IV + Y_{i-1} + X^*$ to the oracle, and receives $Y = E_K(IV + X) = E_K(Y_{i-1} + X^*)$; if $Y = Y_i$, than $X_i = X^*$ is confirmed
- the problem with manipulable IVs
  - if an attacker can directly manipulate the IV (e.g., flip a selected bit of it), then he can make specific changes to the first plaintext block recovered (e.g., flip a selected bit of it)

Generating unpredictable IVs

- IV = $E_K(N)$
  - where N is a nonce (non-repeating value)
  - N may be a counter or a message ID (unique to the message)
  - to ensure non-manipulability, the sender should send N to the receiver (perhaps at the beginning of the CBC encrypted message), who should then compute the IV locally
    - N may be changed by an attacker, but he cannot control the effects made on the value of the IV
- 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!
  - to ensure non-manipulability the sender should send the IV in an encrypted form (e.g., $E_K(IV)$) to the receiver
    - $E_K(IV)$ may be changed, but the attacker cannot control the effects made on the recovered IV
- both approaches also ensure the secrecy of the IV, which is advantageous

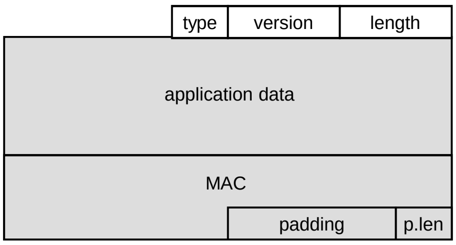
Padding

- 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
- common examples for padding schemes:
  - append a x01 byte and then as many x00 bytes as needed (i.e., 1000…)
  - indicate the length of the padding in the last added byte



- note: padding is 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

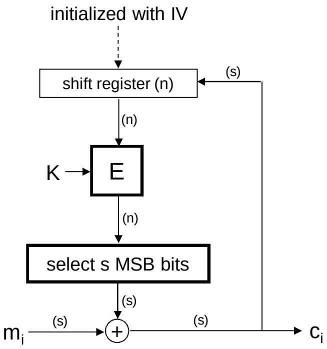
Example: TLS Record Protocol



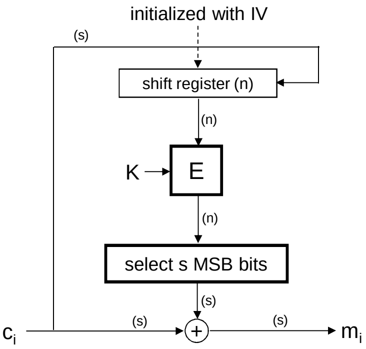
- TLS padding:
  - last byte is the length n of the padding (not including the last byte)
  - all padding bytes have value n
  - examples for correct message tails: x00, x01x01, x02x02x02, …
  - verification: if the last byte is n, then verify if the last n+1 bytes are all n
  - if verification is successful, remove the last n+1 bytes, and proceed with the verification of the MAC

CFB mode

– encrypt



– decrypt



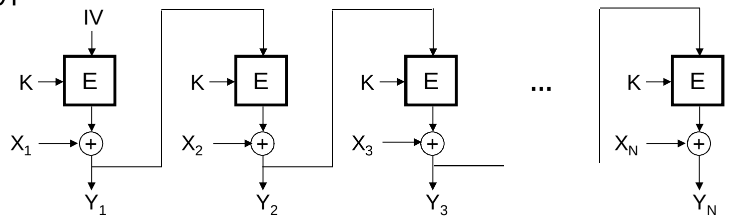
Properties of the CFB mode

- encrypting the same plaintexts under the same key, but different IVs results in different ciphertexts
- ciphertext character c_j depends on m_j and all preceding plaintext characters
  - rearranging ciphertext characters affects decryption
  - proper decryption of a correct ciphertext character requires that the preceding n/s ciphertext characters are correct
- error propagation:
  - one bit error in a ciphertext character c_j has an effect on the decryption of that and the next n/s ciphertext characters (the error remains in the shift register for n/s steps)
    - m_j' has bit errors where C_j had, all the other erroneous plaintext characters are garbage

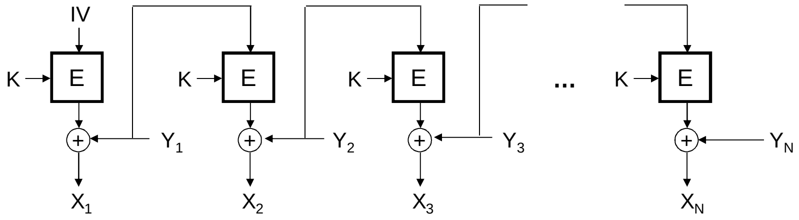
    → an attacker may cause predictable bit changes in the j-th plaintext character !
- self-synchronizing property:
  - recovers from loss of a ciphertext character after n/s steps
- parallel computation (only for decryption), random access, no pre-computation

Another view on CFB

- if s = n, then…
  - encrypt

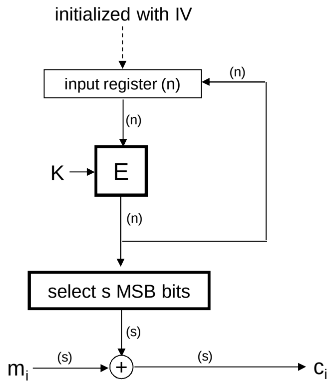


  - decrypt

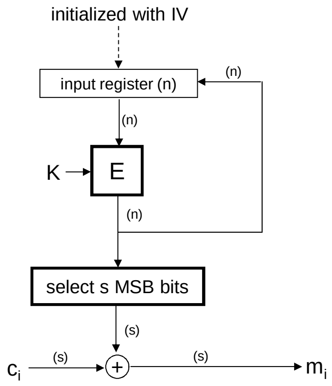


OFB mode

– encrypt



– decrypt

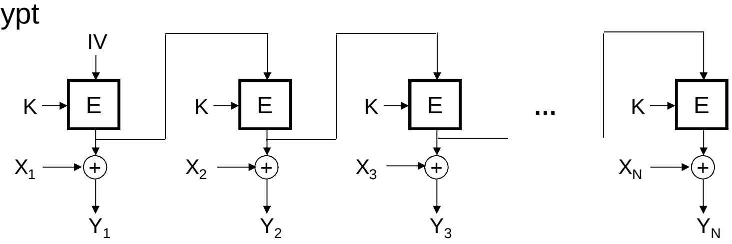


Properties of the OFB mode

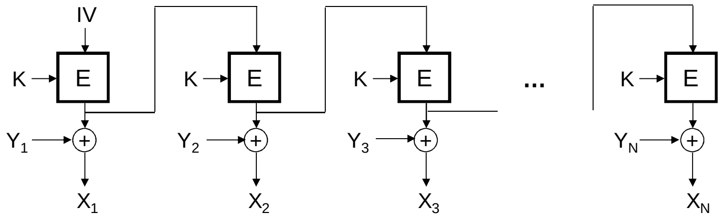
- a different IV should be used for every new message, otherwise messages will be encrypted with the same key stream
- the IV can be sent in clear
  - however, if the IV is modified by the attacker, then the cipher will never recover (unlike CFB)
- ciphertext character c_j depends on m_j only (does not depend on the preceding plaintext characters)
  - however, rearranging ciphertext characters affects decryption
  - statistical properties of the plaintext is hidden due to the random output of the block cipher
- error propagation:
  - one bit error in a ciphertext character c_j has an effect on the decryption of <u>only that ciphertext character</u>
    - m_j' has bit errors where c_j had
    - an attacker may cause predictable bit changes in the j-th plaintext character !!!
- needs synchronization
  - cannot automatically recover from a loss of a ciphertext character
- sequential computation only, no random access, pre-computation is possible

Another view on OFB

- if s = n, then…
  - encrypt

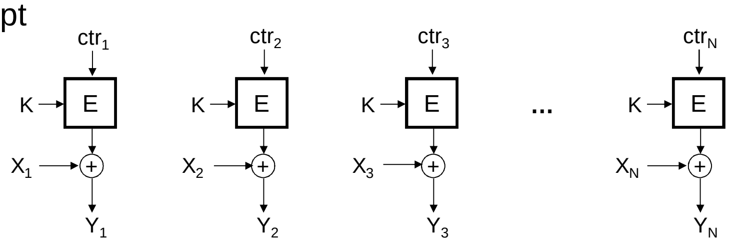


  - decrypt

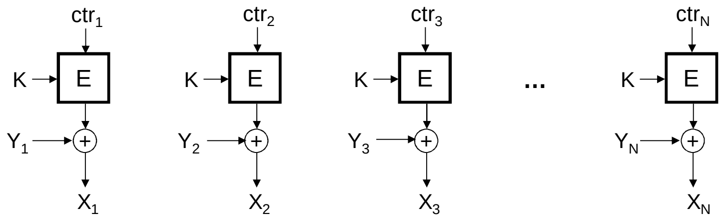


CTR mode

- encrypt



- decrypt



Properties of the CTR mode

- similar to OFB, but …
- parallel computation and random access (unlike OFB), and pre-computation is possible too

Generating counter blocks

- 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 $Y + Y' = X + X'$; if X (or part of it) is known then X' (or part of it) is disclosed to the attacker
- this requires:
  - incrementing function for generating the counter blocks from any initial counter block must ensure that counter blocks do not repeat within a given message
  - the initial counter blocks 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 nonce (e.g., a unique message ID) or it is a 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)

Summary of properties

- 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 and the non-manipulability of the IV is important
  - only the decryption can be parallelized, random access, no pre-computation
  - limited error propagation, self-synchronizing property

- CFB, OFB, CTR:
  - can be used to convert a block cipher into a stream cipher ($s < n$)
    - OFB and CTR → synchronous stream ciphers
    - CFB → self-synchronizing stream-cipher
  - only the encryption algorithm is used, that is why some block ciphers (e.g., Rijndael) are optimized for encryption

---

Summary of properties

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

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

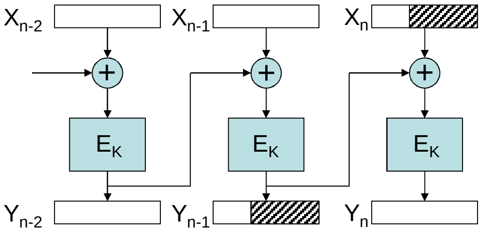
- CTR:
  - non-repeating counters are very important
  - parallelizable, random access, pre-computation
  - no error propagation, needs synchronization

- **none of these modes provide integrity protection !**
- encrypted message is longer than clear message due to padding (except if $s < n$ in CFB, OFB, and CTR modes)

Ciphertext stealing (CTS) in CBC

- encryption:
  - $Y_i = E_K(X_i + Y_{i-1})$ for i = 1..n-1
  - $Y_n = E_K(X_n|0^* + Y_{n-1})$
  - ciphertext: $Y_1 \mid Y_2 \mid \ldots \mid Y_{n-2} \mid Y_n \mid Y_{n-1}^{\mathrm{trunc}(|Xn|)}$

- decryption:
  - $X_i = D_K(Y_i) + Y_{i-1}$ for i = 1..n-2
  - $X_n = D_K(Y_n)^{\mathrm{trunc}(|Xn|)} + Y_{n-1}^{\mathrm{trunc}(|Xn|)}$
  - $Y_{n-1} = D_K(Y_n) + X_n|0^*$
  - $X_{n-1} = D_K(Y_{n-1}) + Y_{n-2}$



Some attacks on CBC

- content leak attack
- cut-and-paste attack
- padding oracle attack

Content leak attack on CBC

- let's assume that we have two encrypted blocks:
  - $Y_i = E_K(X_i + Y_{i-1})$
  - $Y_j = E_K(X_j + Y_{j-1})$

  that happen to be equal:
  - $Y_i = Y_j$
- this means that
  - $D_K(Y_i) = D_K(Y_j)$
  - $X_i + X_j = Y_{i-1} + Y_{j-1}$

- the attacker knows the difference between X_i and X_j
- if X_i (or part of it) is known to the attacker, then X_j (or part of it) is also disclosed

---

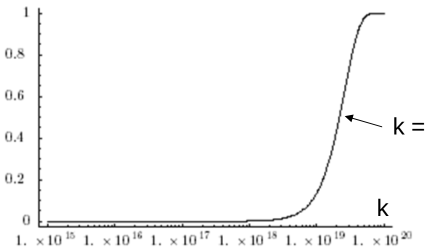
Probability of a matching pair

- Pr{ $Y_i = Y_j$ } = ?

- assume that the block cipher works as a random function
- let P_k be the probability of having no matching pairs among k outputs (size of output space is $N = 2^n$)
  - $P_1 = 1$
  - $P_2 = (N-1)/N$
  - $P_3 = ((N-1)/N)((N-2)/N)$

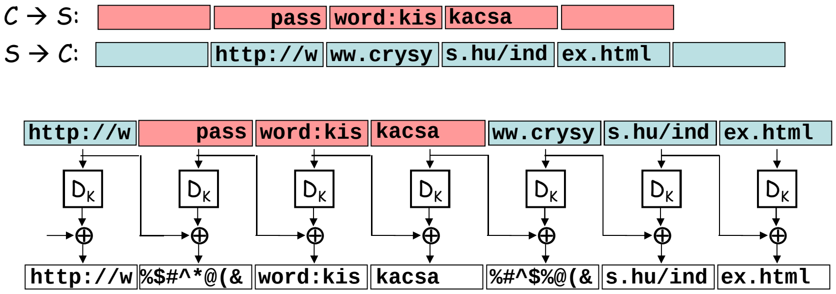
    …
  - $P_k = ((N-1)/N)((N-2)/N) \ldots ((N-k+1)/N) = (1/N^k)(N! / (N-k)!)$

- Pr{ $Y_i = Y_j$ } = $1-P_k$



Cut-and-paste attack on CBC

- given two encrypted messages $Y_1Y_2\ldots Y_p$ and $Y'_1Y'_2\ldots Y'_q$, we can construct $Y_1\ldots Y_i\,Y'_1\ldots Y'_q Y_{i+1}\ldots Y_p$
- this will decrypt into $X_1\ldots X_i\,RX'_2\ldots X'_qR^*X_{i+2}\ldots X_p$
- R and R* are garbage, but the receiver may actually expect random numbers at those positions of the message

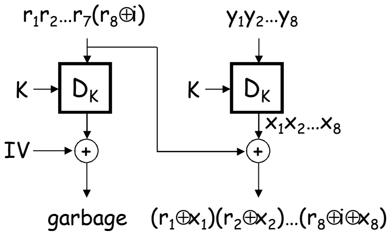


The padding oracle attack on CBC

- padding oracle
  - assume that a system uses CBC encryption/decryption with MAC and padding (in this order!)
  - the receiver of a CBC encrypted message may respond differently in the case of “incorrect padding” and in the case of “correct padding but incorrect MAC”
  - we get 1 bit of information !
- example padding oracle in practice: a TLS server
  - send a random message to a TLS server (chosen ciphertext attack model)
  - the server will drop the message with overwhelming probability
    - either the padding is incorrect (the server responds with a DECRYPTION_FAILED alert)
    - or the MAC is incorrect with very high probability (the server responds with BAD_RECORD_MAC)
- how to exploit this?
  - an attack discovered by Vaudenay in 2002 uses such a padding oracle to decrypt any CBC encrypted message efficiently !
  - vulnerable protocols: SSL/TLS, WTLS, IPsec, …

Recovering the last byte(s)

- assume we have an encrypted block $y_1y_2\ldots y_8 = E_K(x_1x_2\ldots x_8)$
- we want to compute x_8 (the last byte of x)
- idea:
  1. choose a random block $r_1r_2\ldots r_8$; let i = 0
  2. send $r_1r_2\ldots r_7(r_8\oplus i)y_1y_2\ldots y_8$ to the server (oracle)
  3. if there's a padding error, then increment i and go back to step 2
  4. if there's no padding error, then $r\oplus x$ ends with 0 or 11 or 222 …
     - the most likely is that $(r_8\oplus i)\oplus x_8 = 0$, and hence $x_8 = r_8\oplus i$



Recovering the last byte(s)

- assume we get that $x\oplus r$ has a correct padding, but we don't know if it is 0 or 11 or 222 …
- algorithm:
  1. let j = 1
  2. change r_j and send $r_1r_2\ldots r_8y_1y_2\ldots y_8$ to the server again
  3. if the padding is still correct then the j-th byte was not a padding byte; increment j and go back to step 2
  4. if the padding becomes incorrect then the j-th byte was the first padding byte; $x_j \oplus r_j | x_{j+1} \oplus r_{j+1} | \ldots | x_8 \oplus r_8 = (8\text{-}j) |\ldots| (8\text{-}j)$ and hence $x_j\, x_{j+1} \ldots x_8 = r_j\oplus(8\text{-}j)\; r_{j+1}\oplus(8\text{-}j) \ldots r_8\oplus(8\text{-}j)$

x = DE AD BE EF DE AD BE EF
r = 01 23 45 67 DD AE BD EC

$r\oplus x$ = DF 8E FB 88 03 03 03 03

j	r	$r\oplus x$	padding
1	00 23 45 67 DD AE BD EC	DE 8E FB 88 03 03 03 03	OK
2	00 22 45 67 DD AE BD EC	DE 8F FB 88 03 03 03 03	OK
3	00 22 44 67 DD AE BD EC	DE 8F FA 88 03 03 03 03	OK
4	00 22 44 66 DD AE BD EC	DE 8F FA 89 03 03 03 03	OK
5	00 22 44 66 DC AE BD EC	DE 8F FA 89 02 03 03 03	ERROR

$x_5\ x_6\ x_7\ x_8$ = DD⊕03 AE⊕03 BD⊕03 EC⊕03 = DE AD BE EF

Decrypting an entire block

- assume we have an encrypted block $y_1y_2\ldots y_8 = E_K(x_1x_2\ldots x_8)$ and we know the value of $x_jx_{j+1}\ldots x_8$ (using the method for recovering the last byte(s))
- we want to compute x_{j-1}
- algorithm:
  1. choose a random block $r_1r_2\ldots r_8$ such that $r_j = x_j \oplus (9-j)$; $r_{j+1} = x_{j+1} \oplus (9-j)$; … $r_8 = x_8 \oplus (9-j)$;
  2. let i = 0
  3. send $r_1r_2\ldots r_{j-2}(r_{j-1} \oplus i)r_j\ldots r_8y_1y_2\ldots y_8$ to the server (oracle)
  4. if there's a padding error then increment i and go back to step 3
  5. if there's no padding error then $x_{j-1} \oplus r_{j-1} \oplus i = 9-j$ and hence $x_{j-1} = r_{j-1} \oplus i \oplus (9-j)$

```
x = DE AD BE EF DE AD BE EF
r = 01 23 45 67 DA A9 BA EB
r⊕x = DF 8E FB 88 04 04 04 04
i     r                          r⊕x                        padding
0     01 23 45 67 DA A9 BA EB    DF 8E FB 88 04 04 04 04    ERROR
1     01 23 45 66 DA A9 BA EB    DF 8E FB 89 04 04 04 04    ERROR
…     …                                   …                 …
140   01 23 45 EB DA A9 BA EB    DF 8E FB 04 04 04 04 04    OK
```

x_4 = EB⊕04 = EF

Decrypting an entire message

- assume we have a CBC encrypted message (Y_1, Y_2, …, Y_N) where
  - $Y_1 = E_K(X_1 \oplus IV)$
  - $Y_i = E_K(X_i \oplus Y_{i-1})$ (for $1 < i < N$)
  - $Y_N = E_K([X_N|pad|plen] \oplus Y_{N-1})$
- we want to compute X_1, X_2, … X_N
- algorithm:
  - decrypt Y_N using the block decryption method and XOR the result to Y_{N-1}; you get X_N|pad|plen
  - decrypt Y_i using the block decryption method and XOR the result to Y_{i-1}; you get X_i
  - decrypt Y_1 using the block decryption method and XOR the result to IV; you get X_1 (if the IV is secret you cannot get X_1)

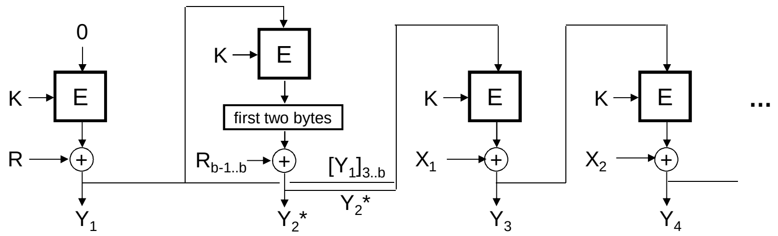
complexity of the whole attack:

on average we need only ½*256*8*N = 1024*N oracle calls !

Lessons learned

- content leak attack → use a sufficiently large block size (e.g., 128 bits)
- cut-and-paste attack → use some integrity protection mechanism (e.g., MAC or authenticated encryption (next lecture))
- padding oracle attack → pay attention on how the MAC function is used (e.g., apply it on the encrypted message)

CFB encryption in OpenPGP



- the receiver verifies if he uses the right key for decryption:
  $[\ E_K(0) + Y_1\ ]_{b-1..b} = [\ E_K(0)\ ]_{b-1..b} + [\ Y_1\ ]_{b-1..b} \ =?\ [\ E_K(Y_1)\ ]_{1..2} + Y_2^*$
- if the above condition holds, then continue decryption, otherwise stop

A chosen ciphertext attack

$[\,E_K(0)\,]_{b-1..b} + [\,Y_1\,]_{b-1..b}$ =? $[\,E_K(Y_1)\,]_{1..2} + Y_2^*$

- we assume that the attacker knows
  - the ciphertext $C_1 \mid C_2^* \mid C_3 \mid C_4 \mid \ldots$
  - the first two bytes of the corresponding plaintext $[\,M_1\,]_{1..2}$
    - note that PGP compresses messages before encrypting them, and the compression method is encoded in the first two bytes of the compressed message
- computing $[\,E_K(0)\,]_{b-1..b}$:
  - send $[\,C_1\,]_{3..b} \mid C_2^* \mid D^* \mid C_3 \mid C_4 \mid \ldots$ to the oracle
  - the oracle verifies if
    $[\,E_K(0)\,]_{b-1..b} + C_2^*$ =? $[\,E_K([C_1]_{3..b} \mid C_2^*)\,]_{1..2} + D^* = [M_1]_{1..2} + [C_3]_{1..2} + D^*$
  - if the oracle accepts the message, then the attacker knows that
    $[\,E_K(0)\,]_{b-1..b} = C_2^* + [M_1]_{1..2} + [C_3]_{1..2} + D^*$
  - otherwise try another D*

A chosen ciphertext attack

$[\,E_K(0)\,]_{b-1..b} + [\,Y_1\,]_{b-1..b}$ =? $[\,E_K(Y_1)\,]_{1..2} + Y_2^*$

- computing $[\,M_2\,]_{1..2}$:
  - send $C_3 \mid D^* \mid C_3 \mid C_4 \mid \ldots$ to the oracle
  - the oracle verifies if
    $[\,E_K(0)\,]_{b-1..b} + [\,C_3\,]_{b-1..b}$ =? $[\,E_K(C_3)\,]_{1..2} + D^*$
  - if the oracle accepts the message, then the attacker knows that
    $[\,E_K(C_3)\,]_{1..2} = [\,E_K(0)\,]_{b-1..b} + [\,C_3\,]_{b-1..b} + D^*$
    $[\,M_2\,]_{1..2} = [\,E_K(C_3)\,]_{1..2} + [\,C_4\,]_{1..2}$
  - otherwise try another D*
- computing $[\,M_3\,]_{1..2}$:
  - send $C_4 \mid D^* \mid C_3 \mid C_4 \mid \ldots$ to the oracle
  - …