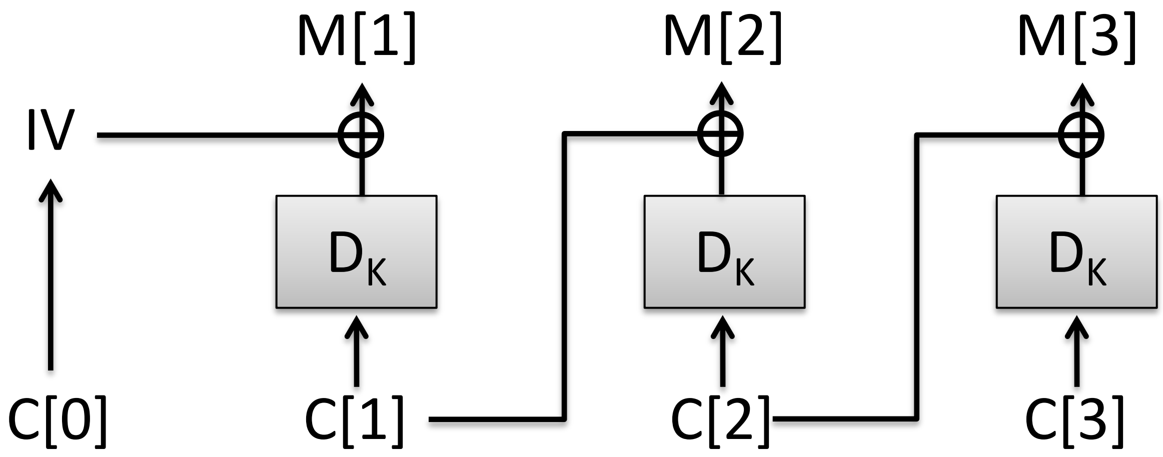
CS 5830
Cryptography

Recap and where we’re at

- IND-CPA secure blockcipher modes of operation
  - CTR mode, CBC mode
  - Message confidentiality under chosen-plaintext attacks
- Limitations of encryption just being IND-CPA
  - Does not provide integrity
  - Does not provide confidentiality under chosen-***ciphertext*** attacks (CCAs)
- Today: confidentiality-breaking CCAs

Decryption errors

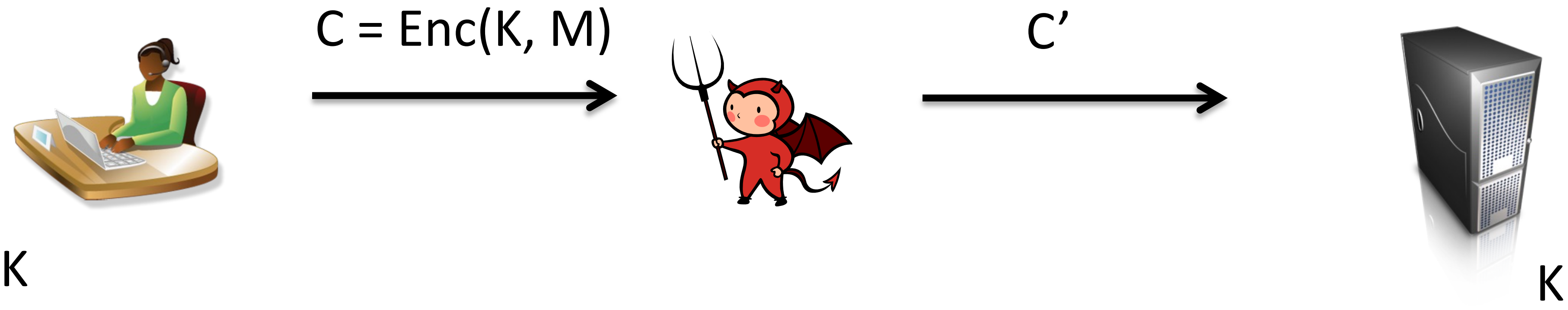


In implementation, what to do when CBC decryption called on bit string that isn't a multiple of n?

Throw an exception or return an error

In pseudocode, cryptographers often denote error case by returning the bottom symbol $\perp$
We'll also say "return error" (same idea)

Integrity attack setting



C = Enc(K, M)

C’

K

K

```
M’ <- Dec(K,C’)
If $M’ \neq \perp$ then
    Pass M’ to application
Return error
```

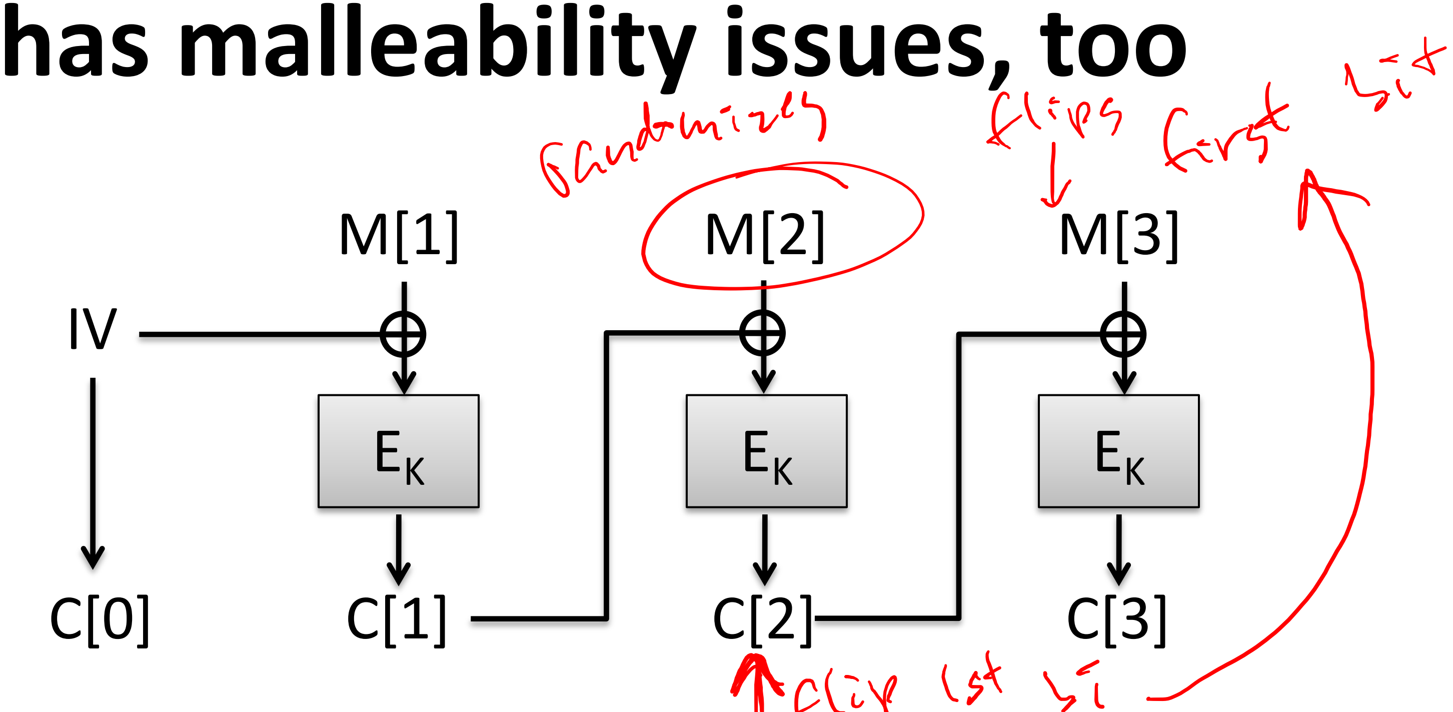
Adversary’s goal:

- get recipient to accept $M’ \neq M$
- compromises integrity of some application

What are some settings where integrity is important?

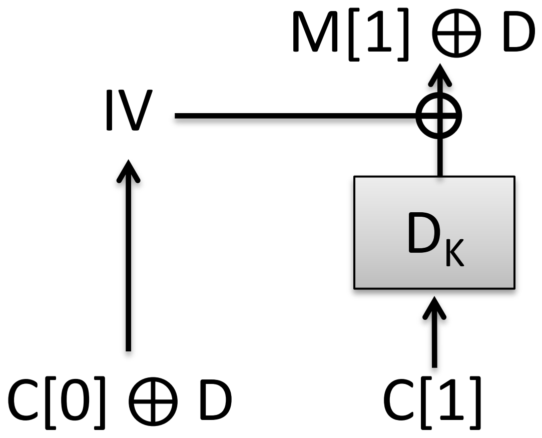
Do any of the schemes we’ve seen so far prevent integrity attacks?

CBC mode has malleability issues, too

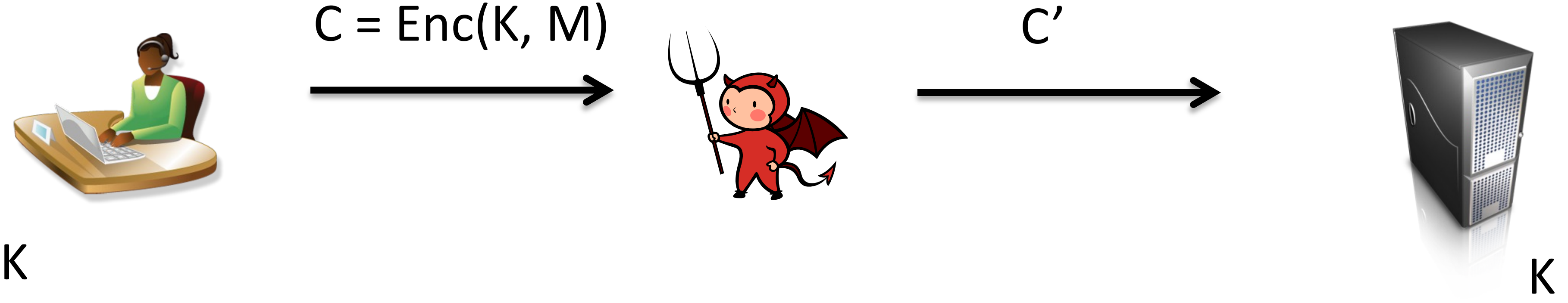


How do we change bits of M received by server?

For any D:



Integrity attack setting



C = Enc(K, M)

C’

K

K

Adversary’s goal:

- get recipient to accept M’ ≠ M
- compromises integrity of some application

```
M’ <- Dec(K,C’)
If M’ ≠ ⊥ then
    Pass M’ to application
Return error
```

Do any of the schemes we’ve seen so far prevent integrity attacks?

All schemes accept any appropriate-length bit string as valid ciphertext!

Padding for CBC mode

- CBC mode handles messages with length a multiple of n bits
- We use padding to make it work for arbitrary message lengths
  - PadCBC, UnpadCBC map to, from strings of length multiple of n

- Padding checks often give rise to chosen-ciphertext attack called ***padding oracle attacks***
  - Given CBC mode encryption C = Enc(K,M) for unknown M
  - Access to oracle that reveals just whether decryption succeeds
  - Recover M

Pseudocode for CBC mode with padding

```
Kg():
K <-$  $\{0,1\}^k$
```

Pick a random key

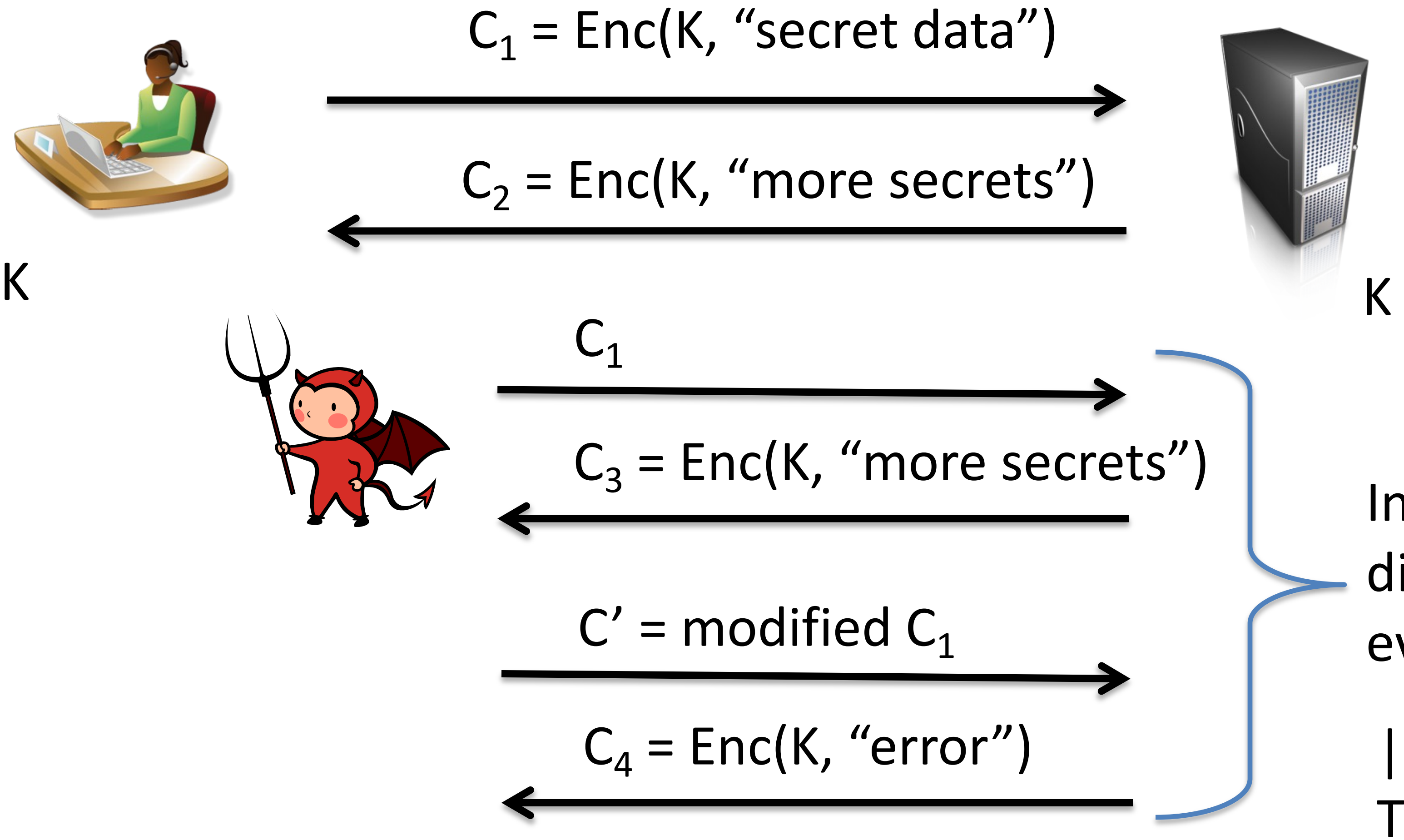
```
CBC-Enc(K,M):
L <- |M|  ; m <- ceil(L/n)
C[0] <- IV <-$ $\{0,1\}^n$
M[1],…,M[m] <-  PadCBC(M,n)
For i = 1 to m do
        C[i] <- $E_K(C[i-1] \oplus M[i])$
Return C[0] || C[1] || … || C[m]
```

PadCBC unambiguously pads M to a sequence of n bit message blocks

```
CBC-Dec(K,C):
For i = 1 to m do
        M[i] <- $C[i-1] \oplus D_K(C[i])$
M  <-  UnpadCBC(M[1],…,M[m],n)
Return M
```

UnpadCBC removes padding, returns appropriately long string
May output ***error*** if padding is wrong
In crypto, errors often denoted by ⊥

Partial decryption oracles arise frequently in practice



C_1 = Enc(K, “secret data”)

C_2 = Enc(K, “more secrets”)

K

K

C_1

C_3 = Enc(K, “more secrets”)

C’ = modified C_1

C_4 = Enc(K, “error”)

In practice usually easy to distinguish C_3 from C_4 even without K

$|C_4| \neq |C_3|$

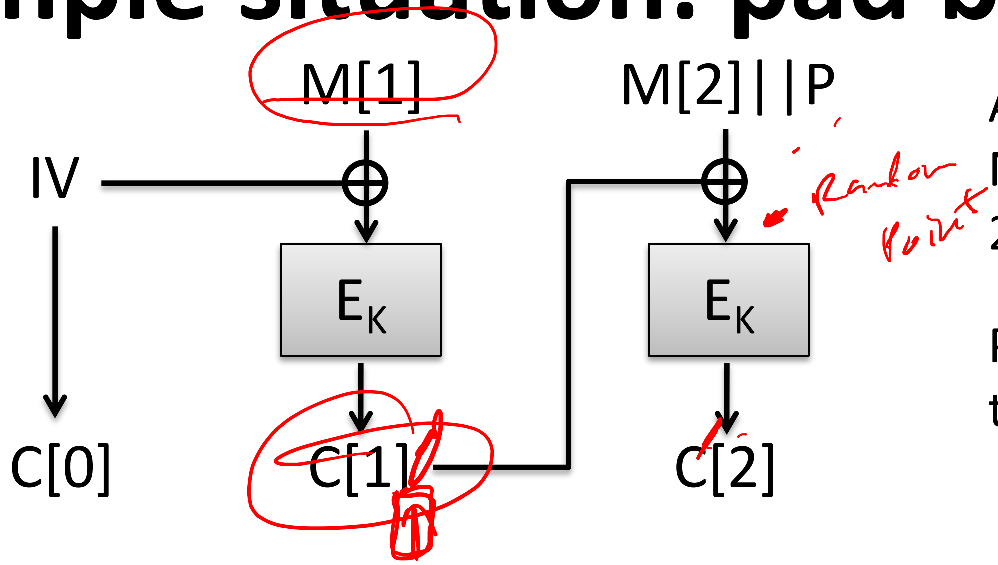
Timing differs for successful vs. unsuccessful decryption

Examples:

TLS/HTTPS partial decryption oracle for network adversaries

Cookies as symmetric ciphertexts for malicious clients

Simple situation: pad by 1 byte



Assume that M[1]||M[2] has length 2n-8 bits

P is one byte of padding that must equal 0x00

C[0] , C[1] , C[2] →

← ok

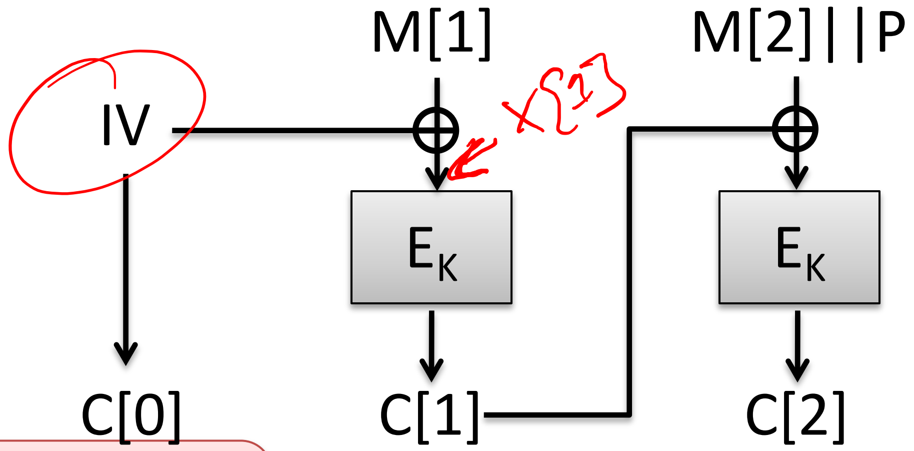
C[0], C[1]⊕1 , C[2] →

← error

Adversary obtains ciphertext C[0],C[1],C[2]

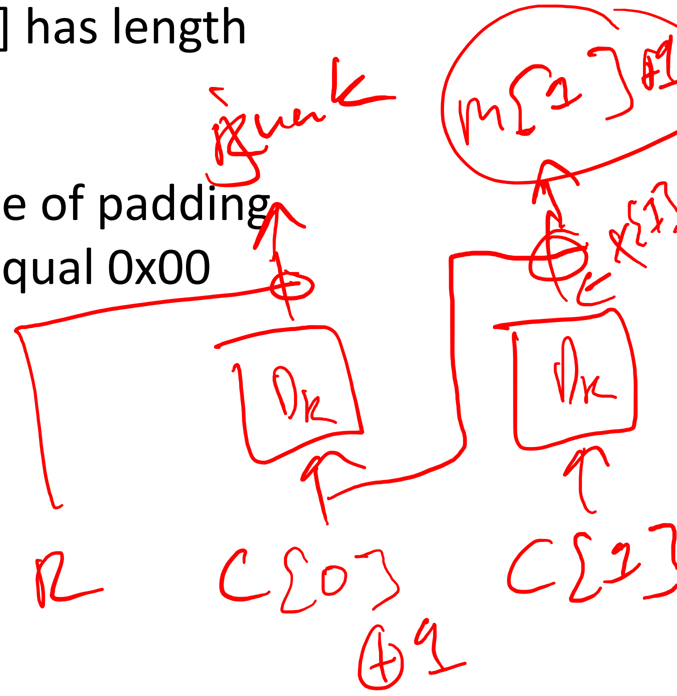
```
Dec(K, C' )
M'[1]||M'[2]||P' = CBC-Dec(K,C')
If P'  ≠ 0x00 then
      Return error
Else
      Return ok
```

Simple situation: pad by 1 byte

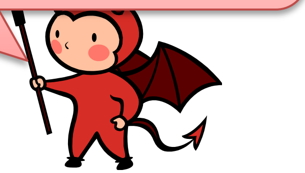


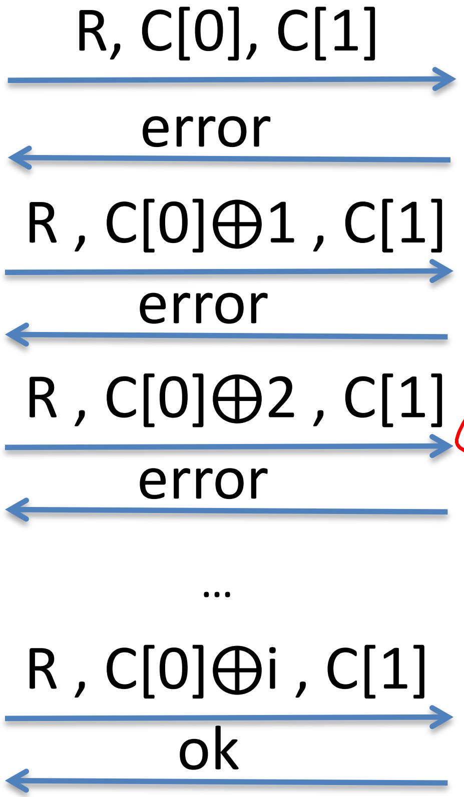
Assume that M[1]||M[2] has length 2n-8 bits

P is one byte of padding that must equal 0x00



Low byte of M1 equals i




R, C[0], C[1]

error

R , C[0]⊕1 , C[1]

error

R , C[0]⊕2 , C[1]

error

…

R , C[0]⊕i , C[1]

ok

```
Dec(K, C' )
M'[1]||M'[2]||P' = CBC-Dec(K,C')
If P' ≠ 0x00 then
    Return error
Else
    Return ok
```

Adversary obtains ciphertext C[0],C[1],C[2]

Let R be arbitrary n bits

PKCS #7 Padding

0x10

0x0F

$$\text{PKCS\#7-Pad}(M) \quad = \quad M \,||\, \underbrace{P \,||\, \ldots \,||\, P}_{\text{P repetitions of byte encoding number of bytes padded}}$$

Possible paddings:

01

02 02

03 03 03

04 04 04 04

…

FF FF FF FF … FF

Hex encodings)

For block length of 16 bytes, don’t need more than 16 bytes of padding (10 10 … 10)

Decryption

(assuming at most one block of padding)

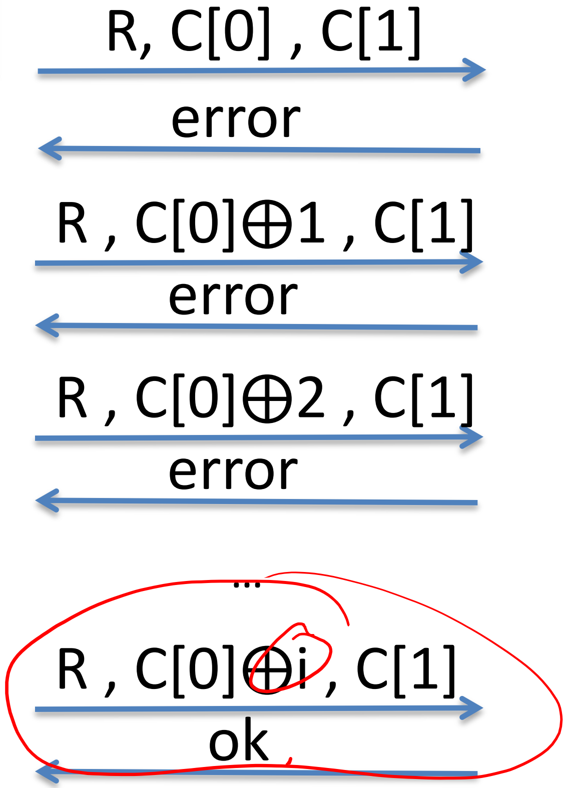
```
Dec( K, C )
M[1] || … || M[m] = CBC-Dec(K,C)
P = RemoveLastByte(M[m])
while i < int(P):
        P' = RemoveLastByte(M[m])
        If P' != P then Return error
        i++
Return ok
```

"ok" / "error" stand-ins for some other behavior:

- Passing data to application layer (web server)
- Returning other error code (if padding fails)

PKCS #7 padding oracles

Low byte of M[1] most likely equals i ⊕ 01



Adversary obtains ciphertext C[0],C[1],C[2]

Let R be arbitrary n bits

```
Dec( K, C )
M'[1] || … || M'[m] = CBC-Dec(K,C)
P = RemoveLastByte(M'[m])
while i < int(P):
        P' = RemoveLastByte(M'[m])
        If P' != P then Return error
        i++
Return ok
```

~~C[2] >> 8~~

Why? Let X[1] = D(K,C1)

$C[0][16] \oplus X[1][16] = M[1][16]$

$C[0][16] \oplus i \oplus X[1][16] = 01$

01 = 02

$M[1][16] \oplus i = 01$

Actually, it could be that:

$M[1][16] \oplus i = 02$

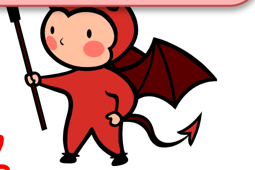
Implies that M[1][15] = 02

We can rule out with an additional query

PKCS #7 padding oracles

Second lowest byte of M[1] equals i ⊕ 02

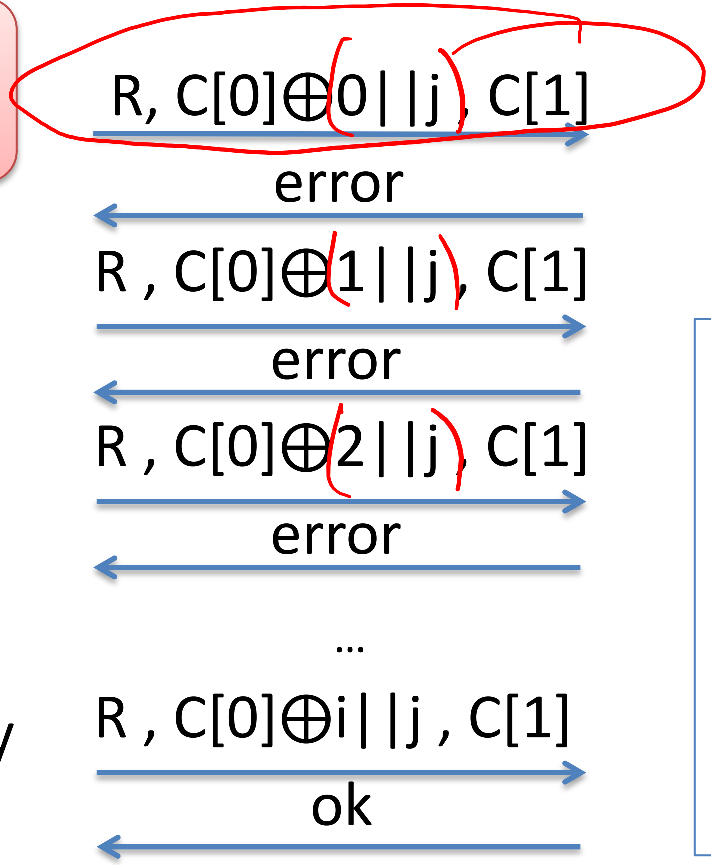
(15)



m[1][15] ⊕ 02 ⊕ 03

Adversary obtains ciphertext C[0],C[1],C[2]

Let R be arbitrary n bits



R, C[0]⊕0||j, C[1]

error

R , C[0]⊕1||j, C[1]

error

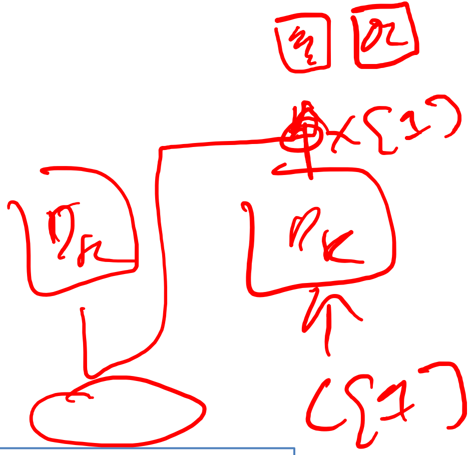
R , C[0]⊕2||j, C[1]

error

…

R , C[0]⊕i||j , C[1]

ok



```
Dec( K, C )
M'[1] || … || M'[m] = CBC-Dec(K,C)
P = RemoveLastByte(M'[m])
while i < int(P):
        P' = RemoveLastByte(M'[m])
        If P' != P then Return error
        i++
Return ok
```

Set j = M[1][16] ⊕ 01 ⊕ 02 3

Keep going to recover entire block of message M[1]

Can repeat with other blocks M[2], M[3], …

Worst case: 256*16 queries per block

(+ check query)

Can we change decryption implementation?

```
Dec( K, C )
M[1] || … || M[m] = CBC-Dec(K,C)
P = RemoveLastByte(M[m])
while i < int(P):
        P’ = RemoveLastByte(M[m])
        If P’ != P then Return error
        i++
Return ok
```

“ok” / “error” stand-ins for some other behavior:

- Passing data to application layer (web server)
- Returning other error code (if padding fails)

Chosen ciphertext attacks against CBC

Attack	Description	Year
Vaudenay	10’s of chosen ciphertexts, recovers message bits from a ciphertext. Called “padding oracle attack”	2001
Canvel et al.	Shows how to use Vaudenay’s ideas against TLS	2003
Degabriele, Paterson	Breaks IPsec encryption-only mode	2006
Albrecht et al.	Plaintext recovery against SSH	2009
Duong, Rizzo	Breaking ASP.net encryption	2011
Jager, Somorovsky	XML encryption standard	2011
Duong, Rizzo	“Beast” attacks against TLS	2011
AlFardan, Paterson	Attack against DTLS	2012
AlFardan, Paterson	Lucky 13 attack against DTLS and TLS	2013
Albrecht, Paterson	Lucky microseconds against Amazon’s s2n library	2016

ASP.NET Attack

- ASP.NET is Microsoft framework for web apps
  - 2010: 25% of all web sites
- “destroy security model of every ASP.NET v4.0 application”
  - Use CBC padding oracle attack to recover secret information
  - Use CBC padding oracle to ***construct ciphertexts***
    - CBC-R technique [Duong, Rizzo]
    - This enables file disclosure vulnerability

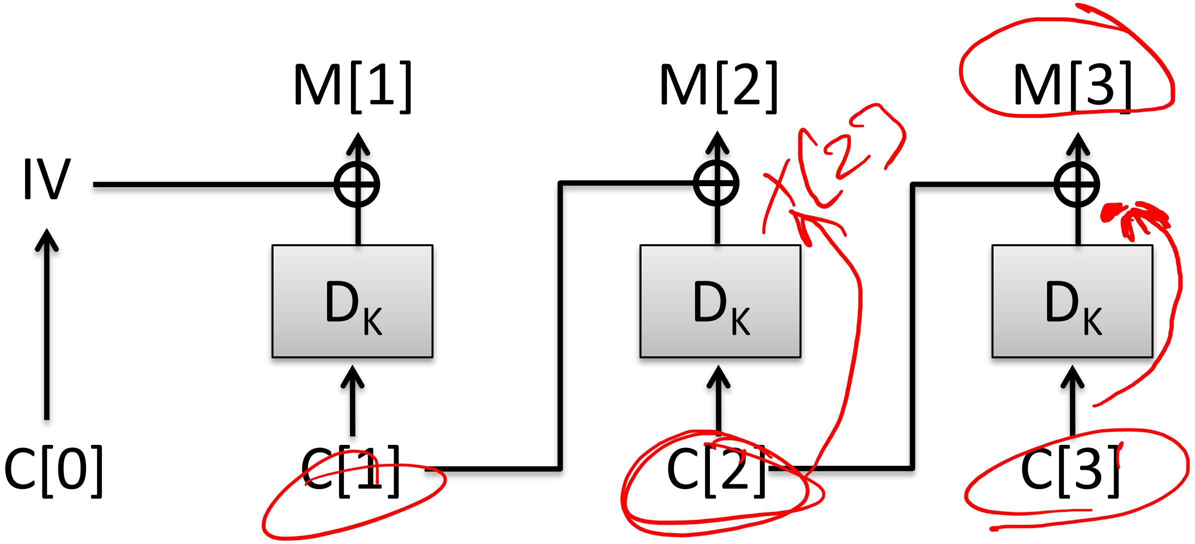
CBC-R approach

Vaudenay’s padding oracle attack gives you access to D_K

Use it to construct ciphertext that decrypts to attacker-controlled message

$m[3] \oplus \underline{x[3]}$

$= \underline{C[2]}$



Use it to construct ciphertext that decrypts to attacker-controlled message

ASP.NET access with WebResource.axd?d=encrypted_id&t=timestamp

Will return file indicated in plaintext under encrypted_id, a CBC ciphertext

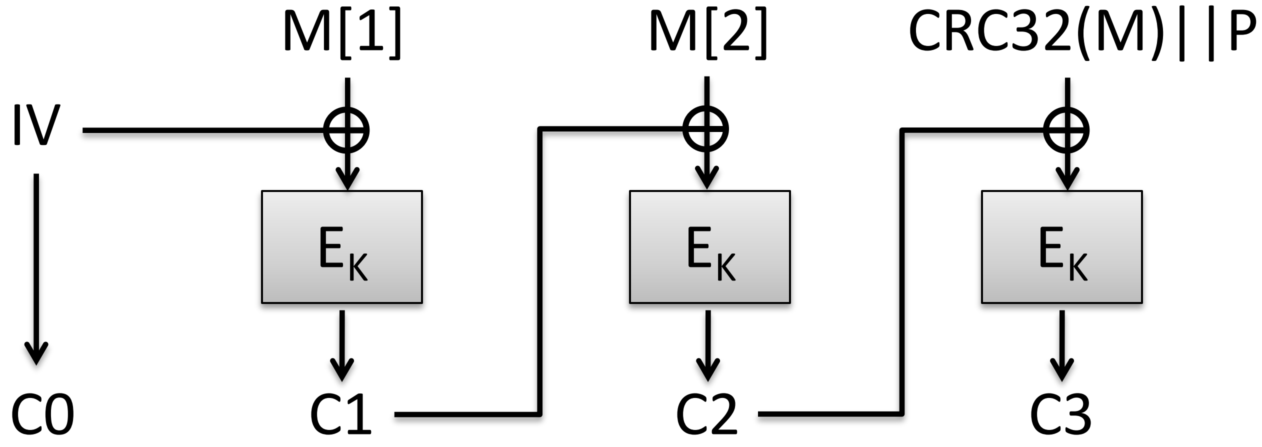
Plaintext format can be: “ R#anything|||~/path/to/file ”

Can use to access web.config file which has all the cryptographic secrets used by ASP.NET instance

Chosen ciphertext attacks against CBC

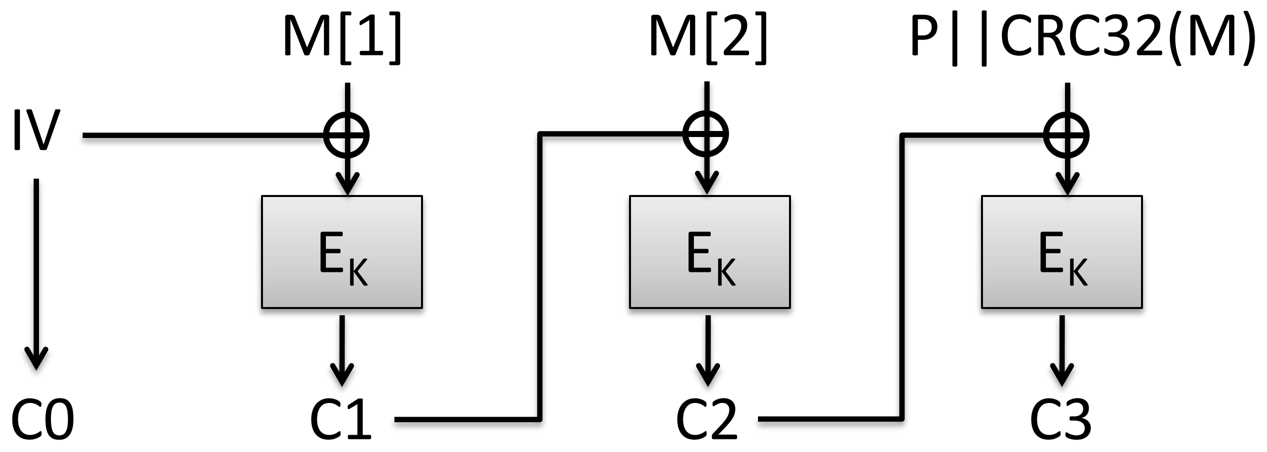
Attack	Description	Year
Vaudenay	10’s of chosen ciphertexts, recovers message bits from a ciphertext. Called “padding oracle attack”	2001
Canvel et al.	Shows how to use Vaudenay’s ideas against TLS	2003
Degabriele, Paterson	Breaks IPsec encryption-only mode	2006
Albrecht et al.	Plaintext recovery against SSH	2009
Duong, Rizzo	Breaking ASP.net encryption	2011
Jager, Somorovsky	XML encryption standard	2011
Duong, Rizzo	“Beast” attacks against TLS	2011
AlFardan, Paterson	Attack against DTLS	2012
AlFardan, Paterson	Lucky 13 attack against DTLS and TLS	2013
Albrecht, Paterson	Lucky microseconds against Amazon’s s2n library	2016

Non-cryptographic checksums?



CRC32(M) is cyclic redundancy code checksum.
Probabilistically catches random errors
Decryption rejects if checksum is invalid

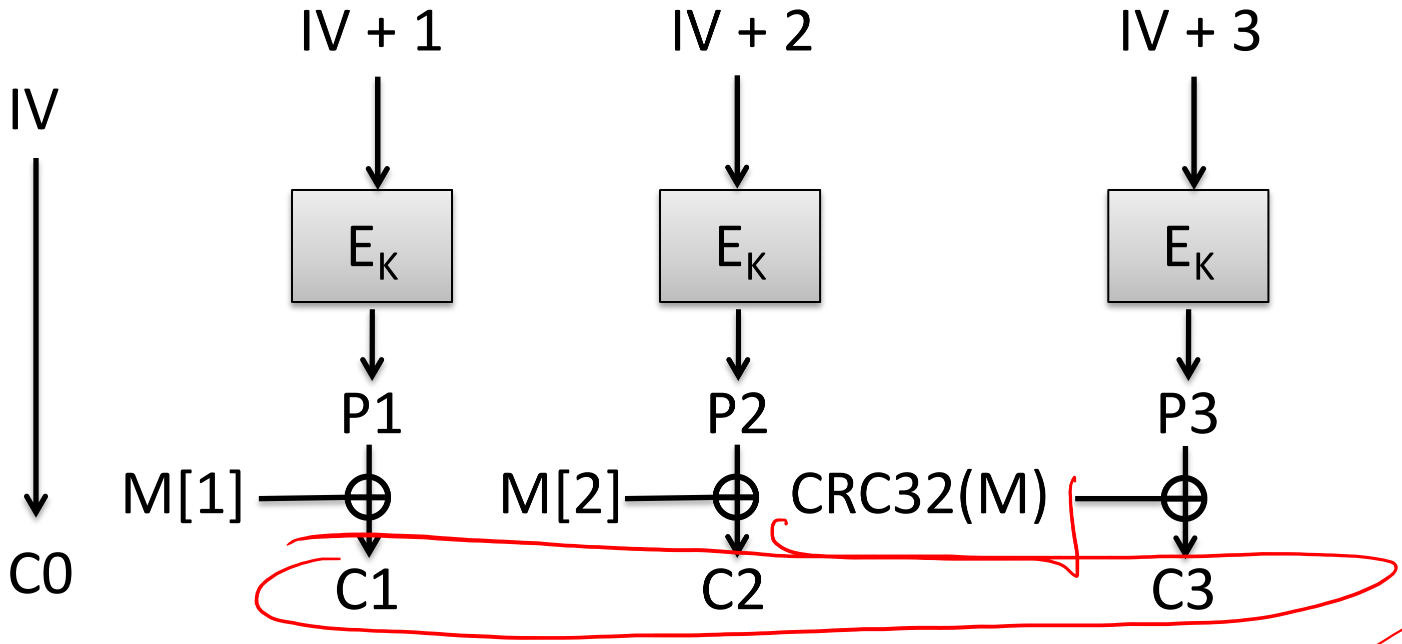
Non-cryptographic checksums?



CRC32(M) is cyclic redundancy code checksum.
Probabilistically catches random errors
Decryption rejects if checksum is invalid

Wagner sketched partial chosen plaintext, chosen ciphertext attack
(see Vaudenay 2002 paper)

Non-cryptographic checksums?



Know: m CRC32(m)

m' CRC32(m')

Can simply maul message and CRC32 checksum to ensure correctness

None of these modes secure for general-purpose encryption

- CTR mode and CBC mode fail in presence of active attacks
  - Cookie example
  - Padding oracle attacks
- Two types of failure:
  - Integrity (trick recipient into accepting wrong message)
  - Confidentiality (padding oracle attacks)
- Need authentication mechanisms to help prevent chosen-ciphertext attacks

Brief digression: need for per-message randomness

- CTR mode uses a per-message random IV
- Deterministic symmetric encryption:
  - Enc(K,M) = Enc(K,M’) iff M = M’
    - In other words, ciphertexts leak (at least) plaintext equality
  - ECB mode is an old, deprecated example. Leaks *much more* than plaintext equality
  - Other examples in wider use that leak just if M = M’:
    - format-preserving encryption (FPE), synthetic IV mode (SIV), …
  - Must be very careful using these, as repetitions can be useful for ***frequency analysis***