CS 5435: Cryptography

Instructor: Tom Ristenpart

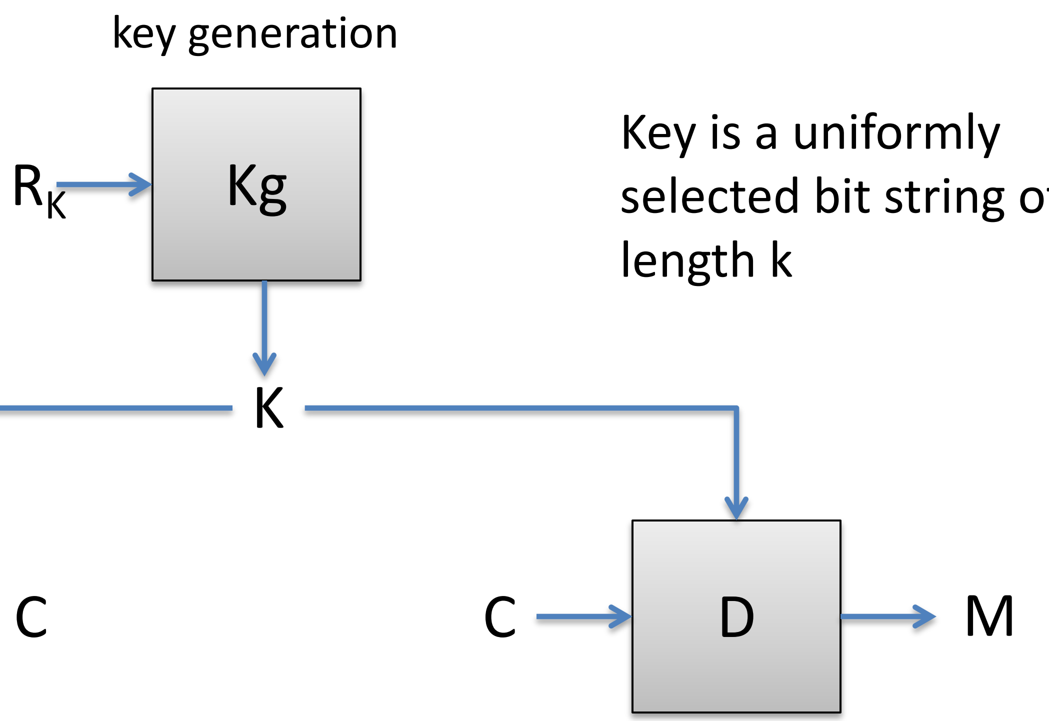
https://github.com/tomrist/cs5435-spring2024

Summary from last time

- Symmetric encryption built now to resist computationally limited adversaries
- Stream ciphers like RC4:
  - Allows reuse of short key (with different IV)
  - Can generate large amounts of output pseudorandom pad
  - Must avoid biases, must use correctly (no nonce repeat)
- Today: block ciphers and modes of operation

Block ciphers

Encryption implements
a family of permutations
on n bit strings,
one permutation for each K



Key is a uniformly
selected bit string of
length k

$E: \{0,1\}^k \times \{0,1\}^n \rightarrow \{0,1\}^n$

Pseudorandom function (PRF) security

F is a random function:

X	Y
00	10
01	11
10	10
11	00

Choose each Y value at random, with replacement

No efficient adversary can distinguish between E_K and random function

- Even given chosen-messages attack: can query X of choosing and get Y, many times

PRF security implies other security goals

F is a random function:

X	Y
00	10
01	11
10	10
11	00

Choose each Y value at random, with replacement

Assume blockcipher E is secure PRF. Can an adversary:

- recover M given $E_K(M)$ for large, random M?
- recover K given M and $E_K(M)$?
- Distinguish between $E_K(M)$ and $E_K(M')$ for $M \neq M'$?

If blockcipher does not resist any of attacks 1, 2, 3, can it be a good PRF?

Data encryption standard (DES)

Originally called Lucifer

- team at IBM
- input from NSA
- standardized by NIST in 1976

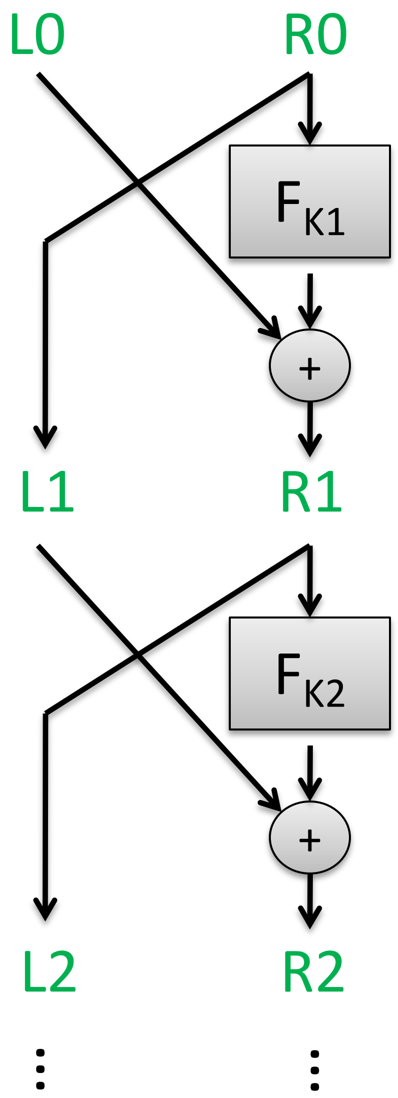
$n = 64$
$k = 56$

Number of keys:
72,057,594,037,927,936

Split 64-bit input into L0,R0 of 32 bits each

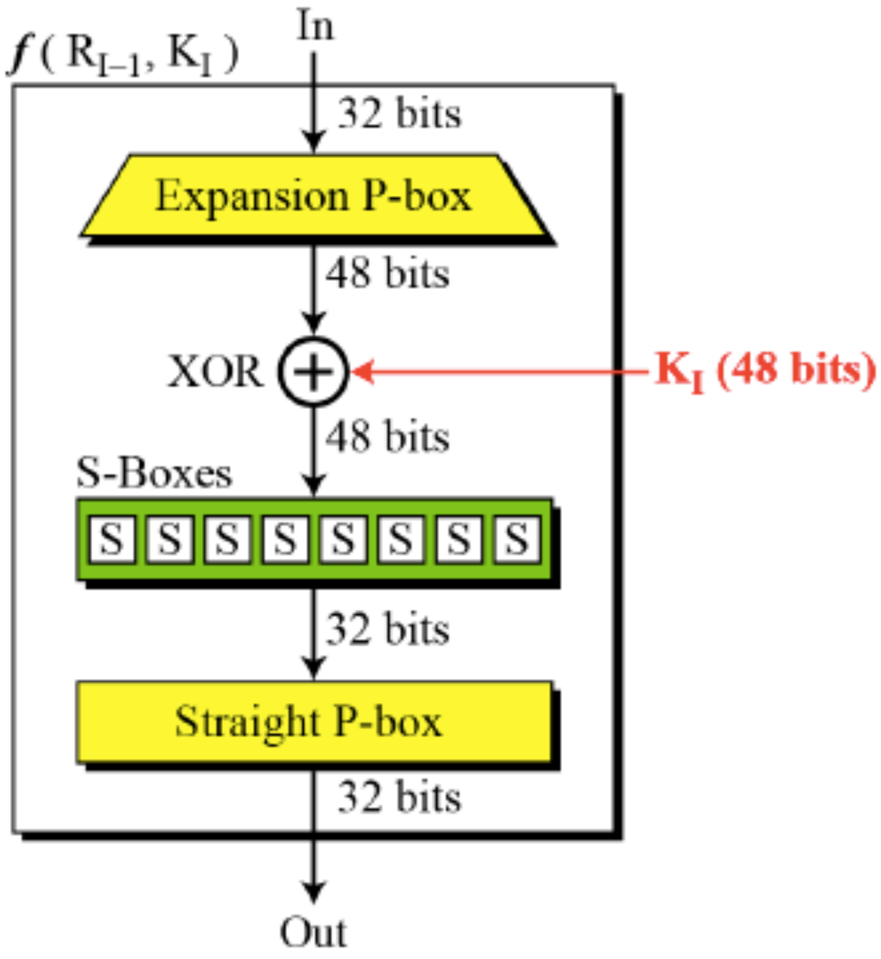
Repeat Feistel round 16 times

Each round applies function F using separate round key



DES round functions

- P-box expands 32 bits to 48 bits and permutes
- S-boxes: 6-bit to 4-bit lookup tables
- XOR in round key
  - 16 48-bit round keys derived via key schedule from 56 bit key deterministically
- How S-boxes chosen? Why particular permutations?
  - Resist cryptanalytic attacks known to NSA at the time (discovered later in 1990s)
  - Differential cryptanalysis



Best attacks against DES

Attack	Attack type	Complexity	Year
Biham, Shamir	Chosen plaintexts, recovers key	2^{47} plaintext, ciphertext pairs	1992
Matsui	Known plaintext, ciphertext pairs, recovers key	2^{42} plaintext, ciphertext pairs, ~2^{41} DES computations	1993
DESCHALL	Unknown plaintext, recovers key	$2^{56/4}$ DES computations 41 days	1997
EFF Deepcrack	Unknown plaintext, recovers key	~4.5 days	1998
Deepcrack + DESCHALL	Unknown plaintext, recovers key	22 hours	1999

- DES is still used in some places
- 3DES (use DES 3 times in a row with more keys) expands keyspace and still used widely in practice

Advanced Encryption Standard (AES)

Response to 1999 attacks:

- NIST has design competition for new block cipher standard
- 5 year design competition
- 15 designs, Rijndael design chosen

Advanced Encryption Standard (AES)

A form of key-alternating cipher

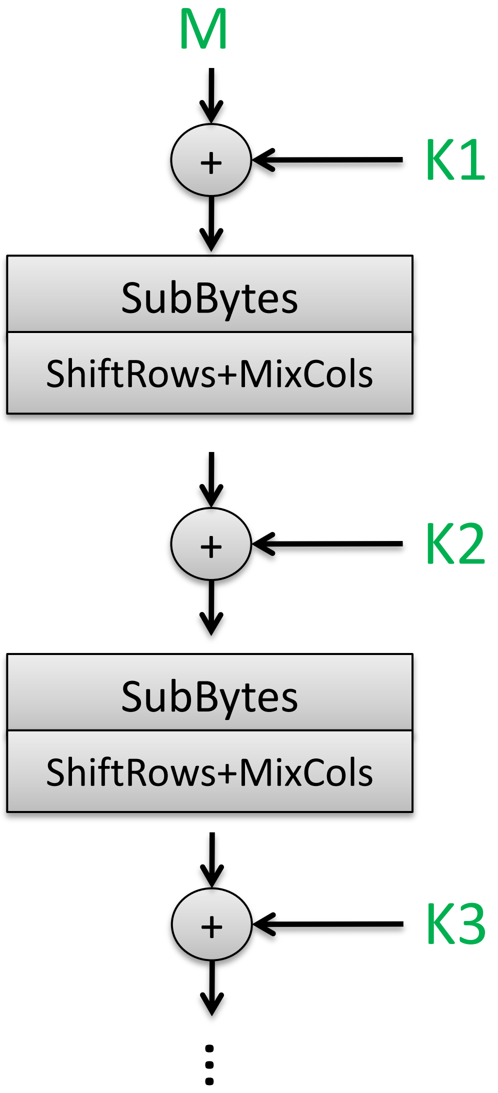
n = 128
k = 128, 192, 256

Number of keys for k=128:
340,282,366,920,938,463,463,374,607,431,768,211,456

Substitution-permutation design.
For k=128 uses 10 rounds of:

1) SubBytes (non-linear 8-bit S-boxes)
2) ShiftRows & MixCols (linear permutation)
3) XOR'ing in a round key

(Last round skips MixCols)

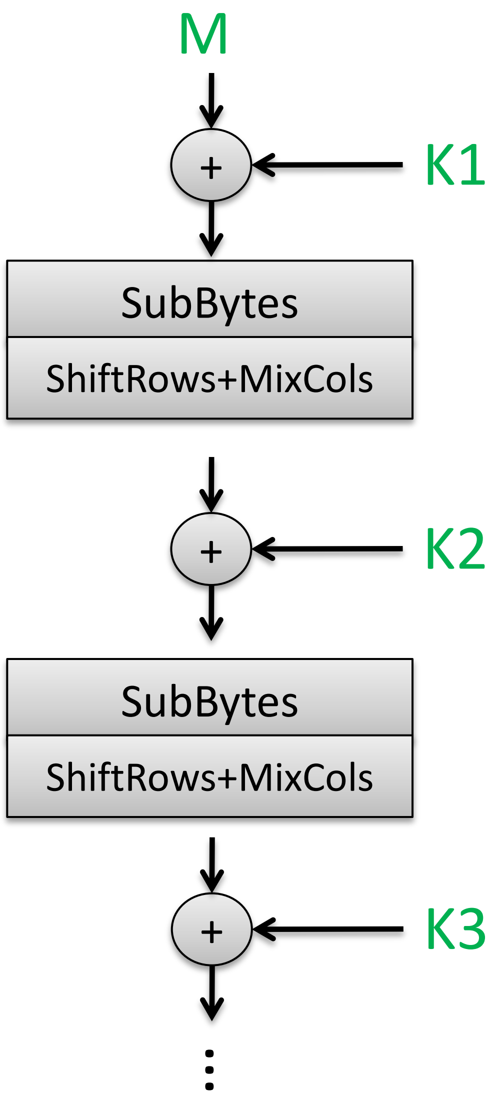


Advanced Encryption Standard (AES)

Designed to resist linear & differential cryptanalysis

“Wide-trail” strategy

- Ensure large # of Sboxes involved in multi-round trail (sequence of intermediate state bits)
- Use coding theory viewpoint to build permutations to ensure rapid ***diffusion***



Best attacks against AES

Attack	Attack type	Complexity	Year
Bogdanov, Khovratovich, Rechberger	chosen ciphertext, recovers key	$2^{126.1}$ time + some data overheads	2011

- Brute force requires time at most 2^{128}
- Approximately factor 4 speedup

AES design still considered a secure PRF
Must implement securely (e.g., AES-NI)

Block cipher summary

- Blockciphers and their security goals
  - Assume good blockciphers that achieve PRF security up to implications of best-known generic attacks
  - ~ 2^k time (exhaustive key search)
  - ~ $2^{n/2}$ time (birthday attacks)

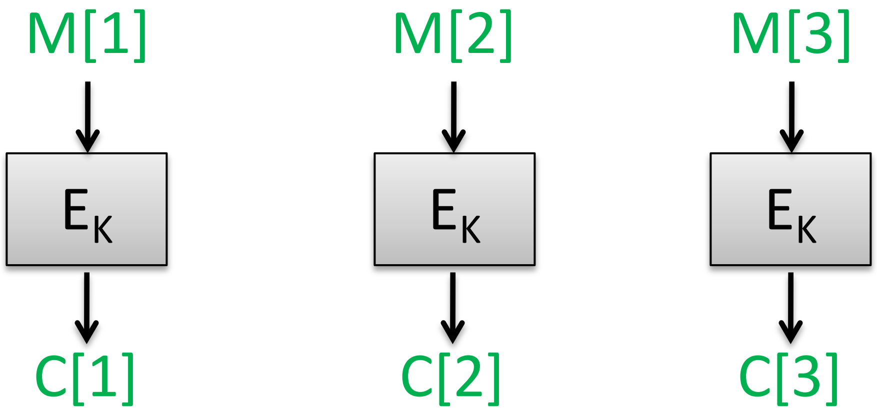
- Today: modes of operation, IND-CPA security, & (time allowing) chosen-ciphertext attacks

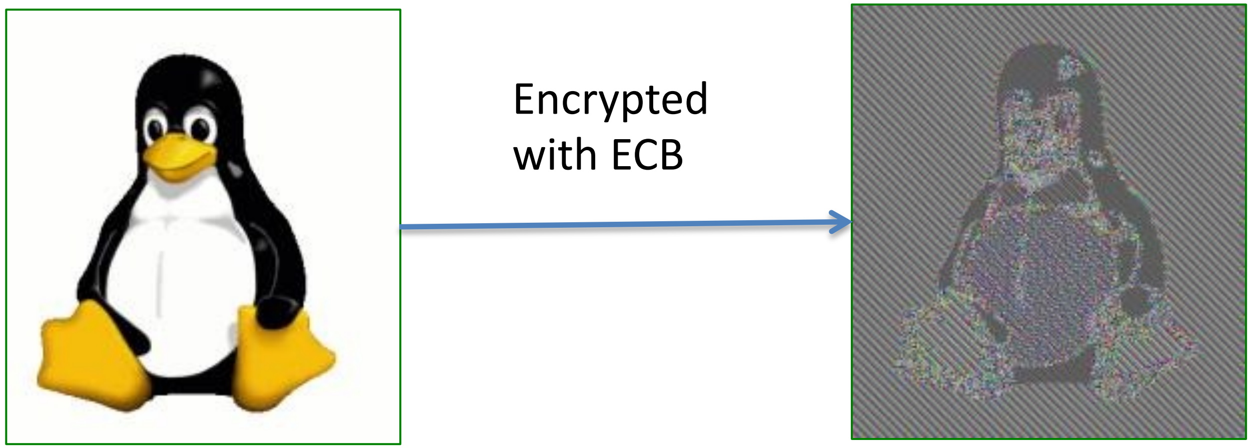
Block cipher modes of operation

Electronic codebook (ECB) mode

Pad message M to M[1],M[2],M[3],... where each block M[i] is n bits

Then:

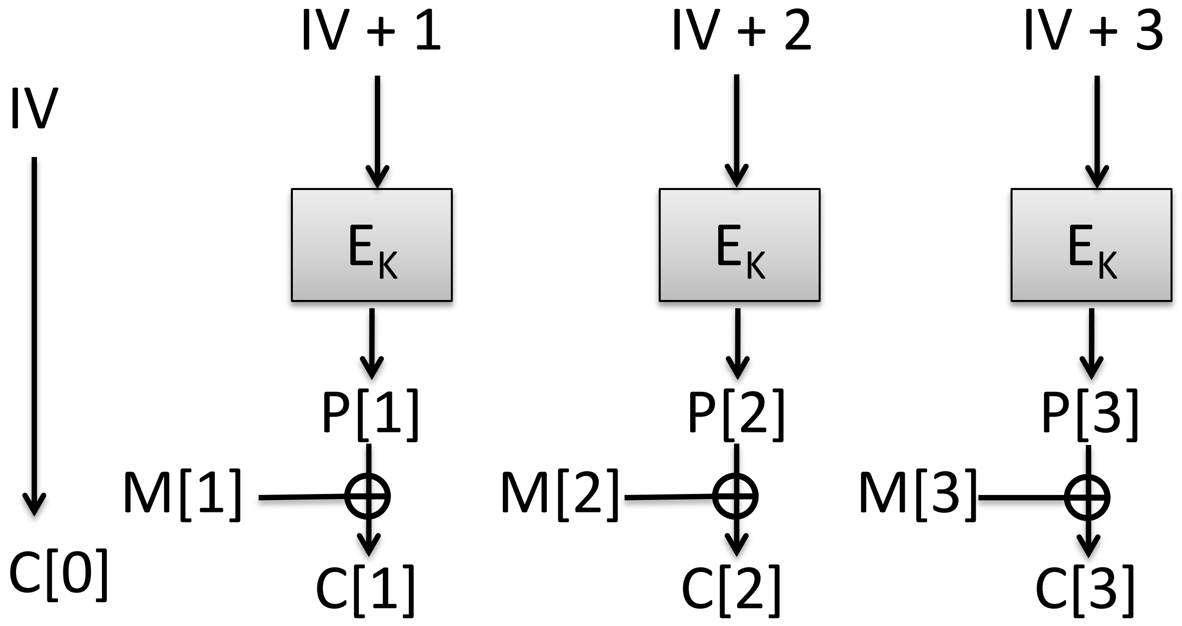




Images courtesy of
http://en.wikipedia.org/wiki/Block_cipher_modes_of_operation

CTR mode: build a stream cipher form block cipher

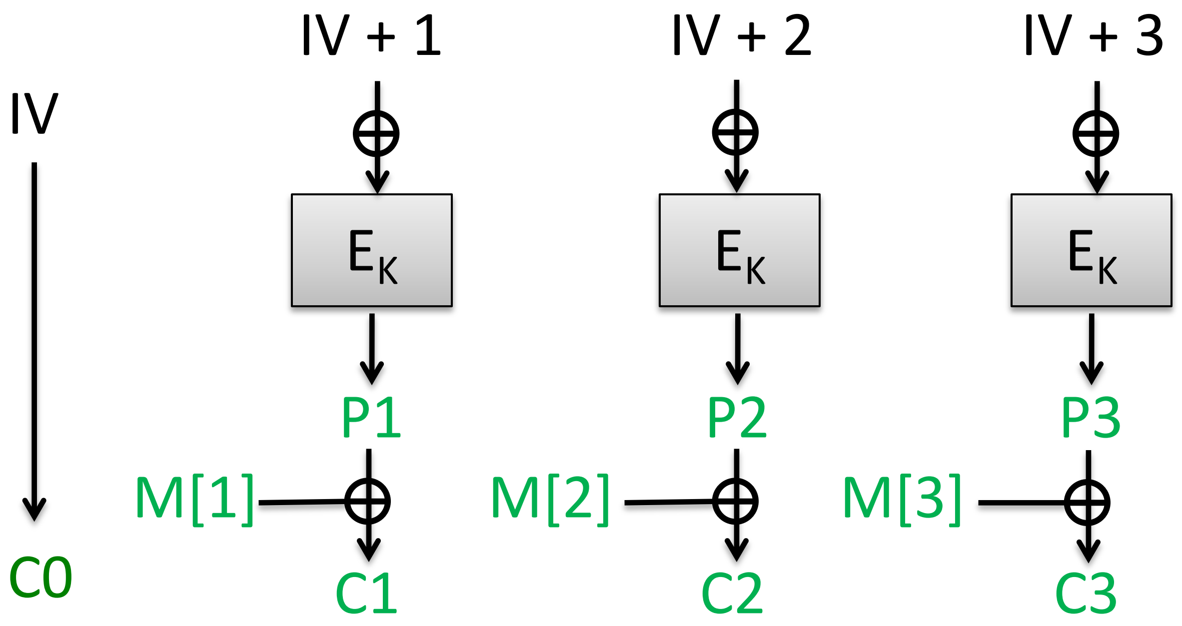
Block cipher E : $\{0,1\}^k \times \{0,1\}^n \rightarrow \{0,1\}^n$ is a family of permutations
Should be secure as a pseudorandom function (PRF)



CTR mode provides message confidentiality (nothing about message from ciphertext) assuming E is a PRF and number of message blocks encrypted << $2^{n/2}$

ECB, CTR: blockcipher modes of operation

- How do we encrypt long messages with block cipher?
  - Modes of operation
- Long history: NIST standard
  - First published in 1980 specifically for DES
  - 2001 version: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38a.pdf
- Analyses starting in 1990s for chosen-plaintext attacks
- Analyses starting in 2000s for chosen-ciphertext attacks



Can attacker learn K from just C0,C1,C2,C3?

Implies attacker can break E, i.e. recover block cipher key

Can attacker learn M = M[1],M[2],M[3] from C0,C1,C2,C3?

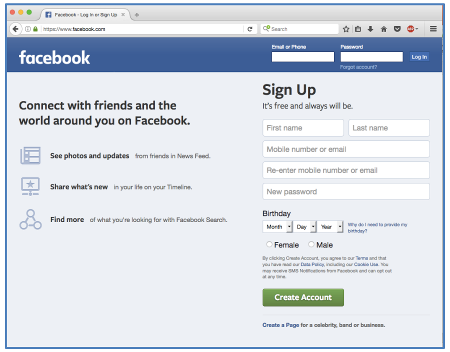
Implies attacker can break PRF security of E

Can attacker learn one bit of M from C0,C1,C2,C3?

Implies attacker can break PRF security of E

Passive adversaries cannot learn anything about messages

Session handling and login



GET /index.html

Set-Cookie: AnonSessID=134fds1431

POST /login.html?name=bob&pw=12345

Cookie: AnonSessID=134fds1431

Set-Cookie: SessID=83431Adf

GET /account.html

Cookie: SessID=83431Adf

Security problems here?

POST /login.html?name=bob&pw=12345

Cookie: AnonSessID=134fds1431

Set-Cookie: SessID=83431Adf

Secret key K only known to server

GET /account.html

Cookie: SessID=83431Adf

83431Adf = CTR-Enc(K, “admin=0”)

Malicious client can simply flip a few bits to change admin=1

Example of an ***integrity / authenticity violation***

Soon we will build authentication mechanisms to prevent this

NIST Modes

- **Electronic codebook mode (ECB)**
- **Counter mode (CTR)**
- **Ciphertext block chaining mode (CBC)**
- Ciphertext feedback mode (CFB)
- Offset feedback mode (OFB)

CTR, CBC found widespread use

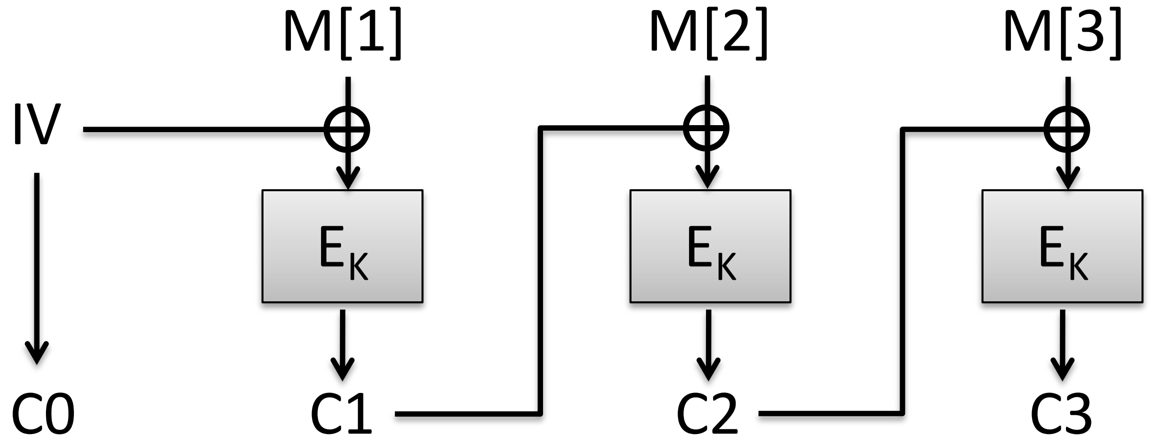
CBC mode

Ciphertext block chaining (CBC)

Pad message M to M[1],M[2],M[3],... where each block M[i] is n bits

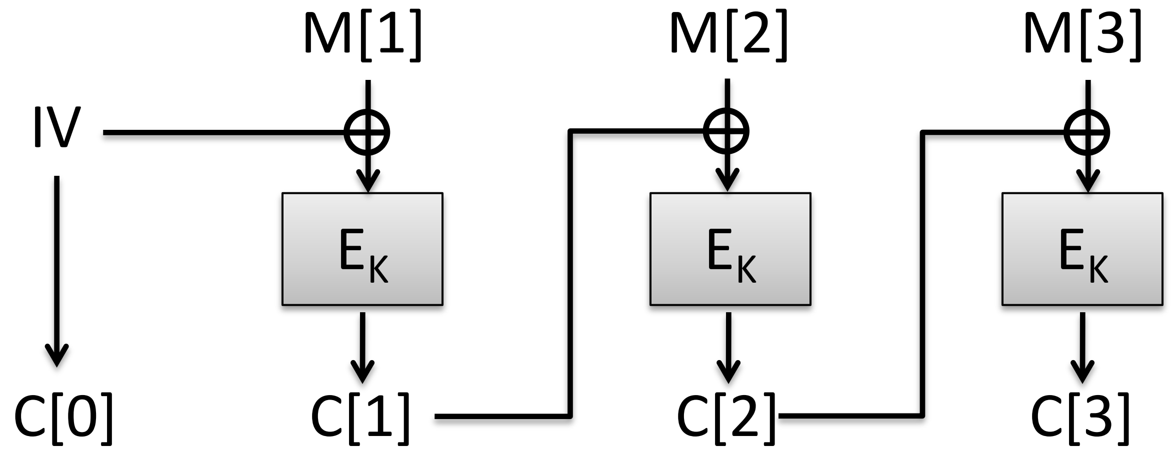
Choose random n-bit string IV

Then:



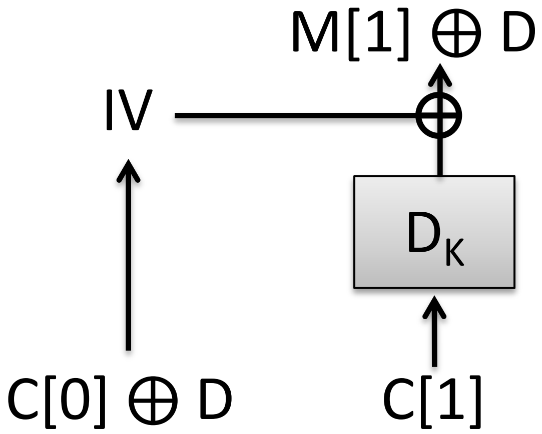
How do we decrypt?

CBC mode has malleability issues, too



How do we change bits of M received by server?

For any D:



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
  - Will specify example padding schemes later

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 $\perp$

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
  - Will specify example padding schemes later
- 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

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”)

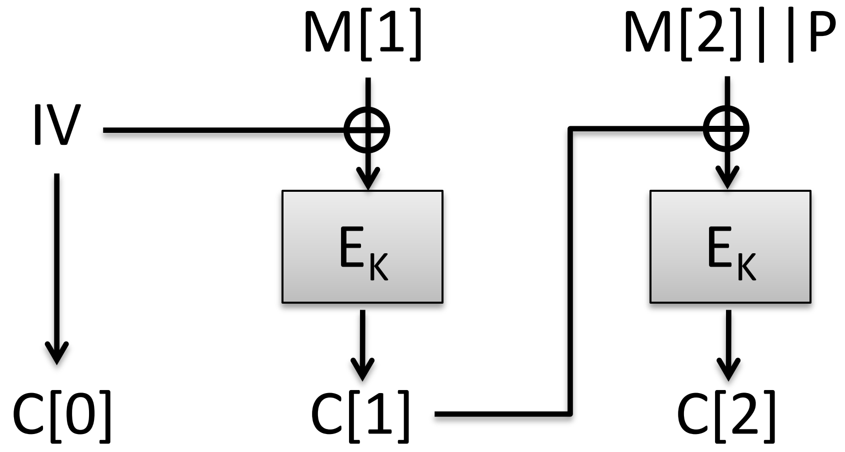
In practice usually easy to distinguish C_3 from C_4 even without K

$|C_4| \neq |C_3|$

Timing differs for successful vs. unsuccessful decryption

TLS/HTTPS canonical examples where decryption oracles arise

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

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

C[0] , C[1] , C[2] →

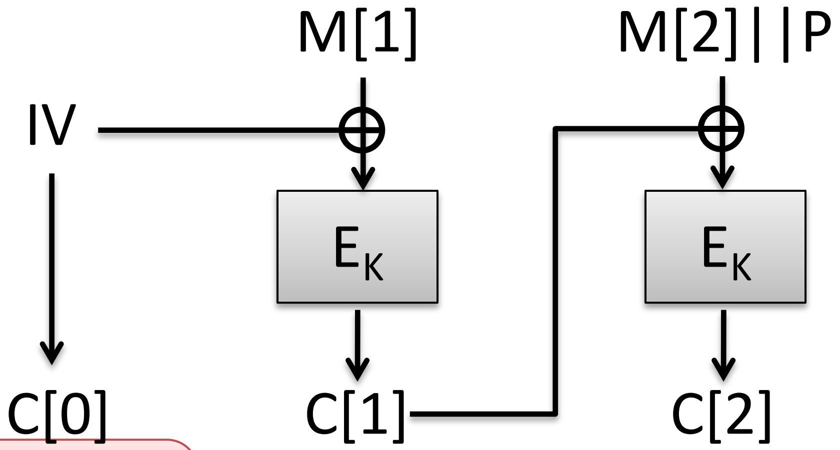
← ok

C[0], C[1]⊕1 , C[2] →

← error

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

Adversary obtains ciphertext C[0],C[1],C[2]
Let R be arbitrary n bits

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

<u>Dec(K, C’)</u>
M’[1]||M’[2]||P’ = CBC-Dec(K,C’)
If P’ ≠ 0x00 then
    Return error
Else
    Return ok

PKCS #7 Padding

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

Possible paddings:

01
02 02
03 03 03
04 04 04 04
…
FF FF FF FF … FF

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
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 $i \oplus 01$

R, C[0] , C[1]

error

R , $C[0]\oplus 1$, C[1]

error

R , $C[0]\oplus 2$, C[1]

error

…

R , $C[0]\oplus i$, C[1]

ok

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

Let R be arbitrary n bits

<u>Dec(K, C)</u>

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

Return ok

<u>**Why?**</u> Let X[1] = D(K,C1)

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

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

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

Actually, it could be that:

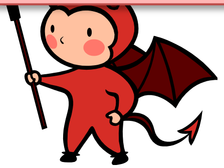
$M[1][16] \oplus i = 02$

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



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
Return ok
```

Set j = M[1][16] ⊕ 01 ⊕ 02

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

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
Return ok
```

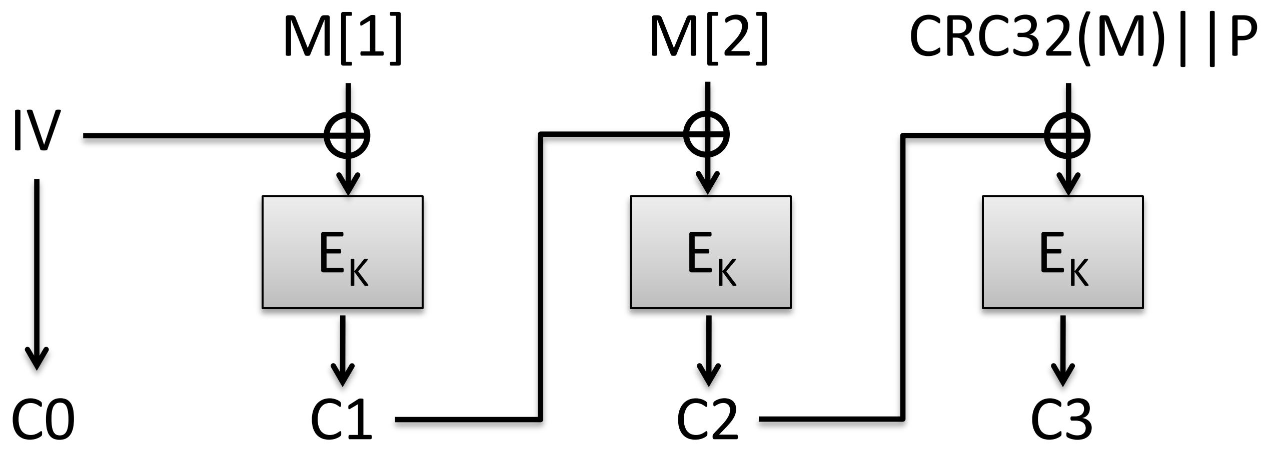
"ok" / "error" stand-ins for some other behavior:

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

Chosen ciphertext attacks against CBC

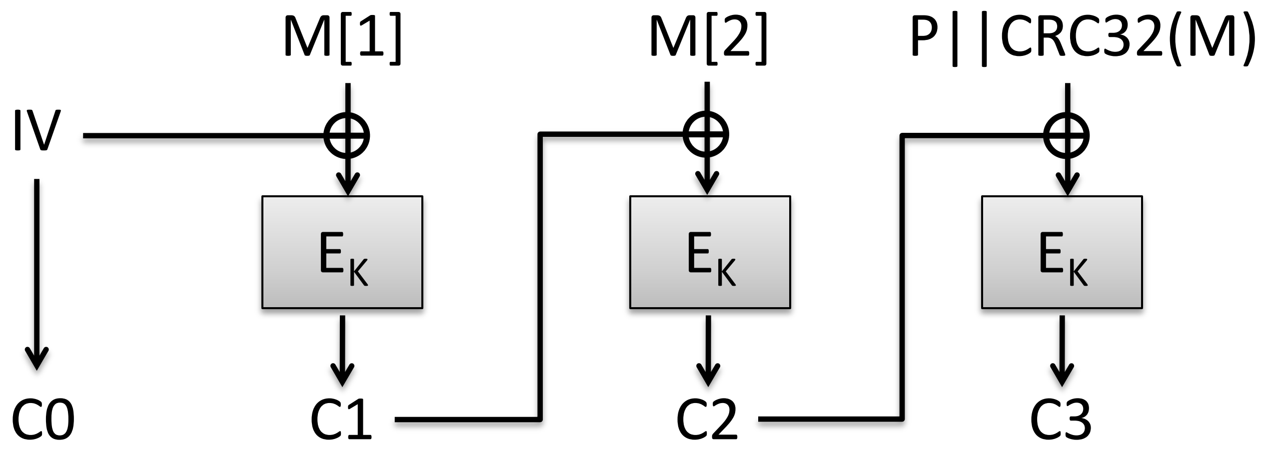
Attack	Description	Year
Vaudenay	10’s of chosen ciphertexts, recovers message bits from a ciphertext. Called “padding oracle attack”	2001
Canvel et al.	Shows how to use Vaudenay’s ideas against TLS	2003
Degabriele, Paterson	Breaks IPsec encryption-only mode	2006
Albrecht et al.	Plaintext recovery against 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

Non-cryptographic checksums?



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

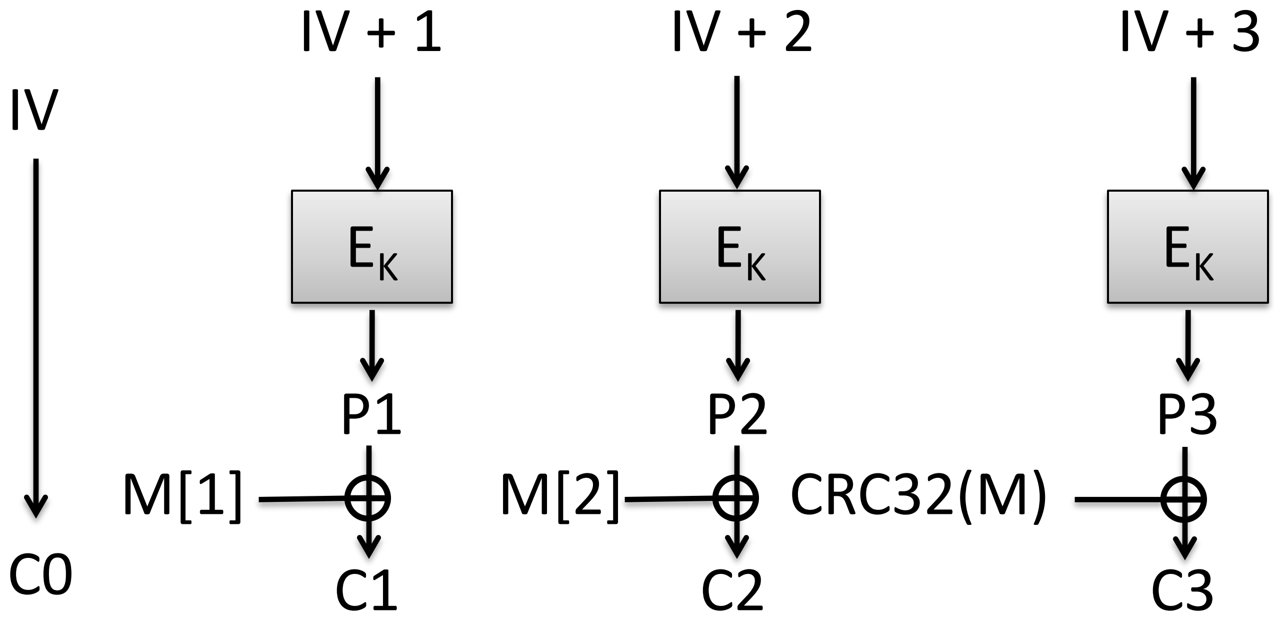
Non-cryptographic checksums?



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?



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

- Need authentication mechanisms to help prevent chosen-ciphertext attacks