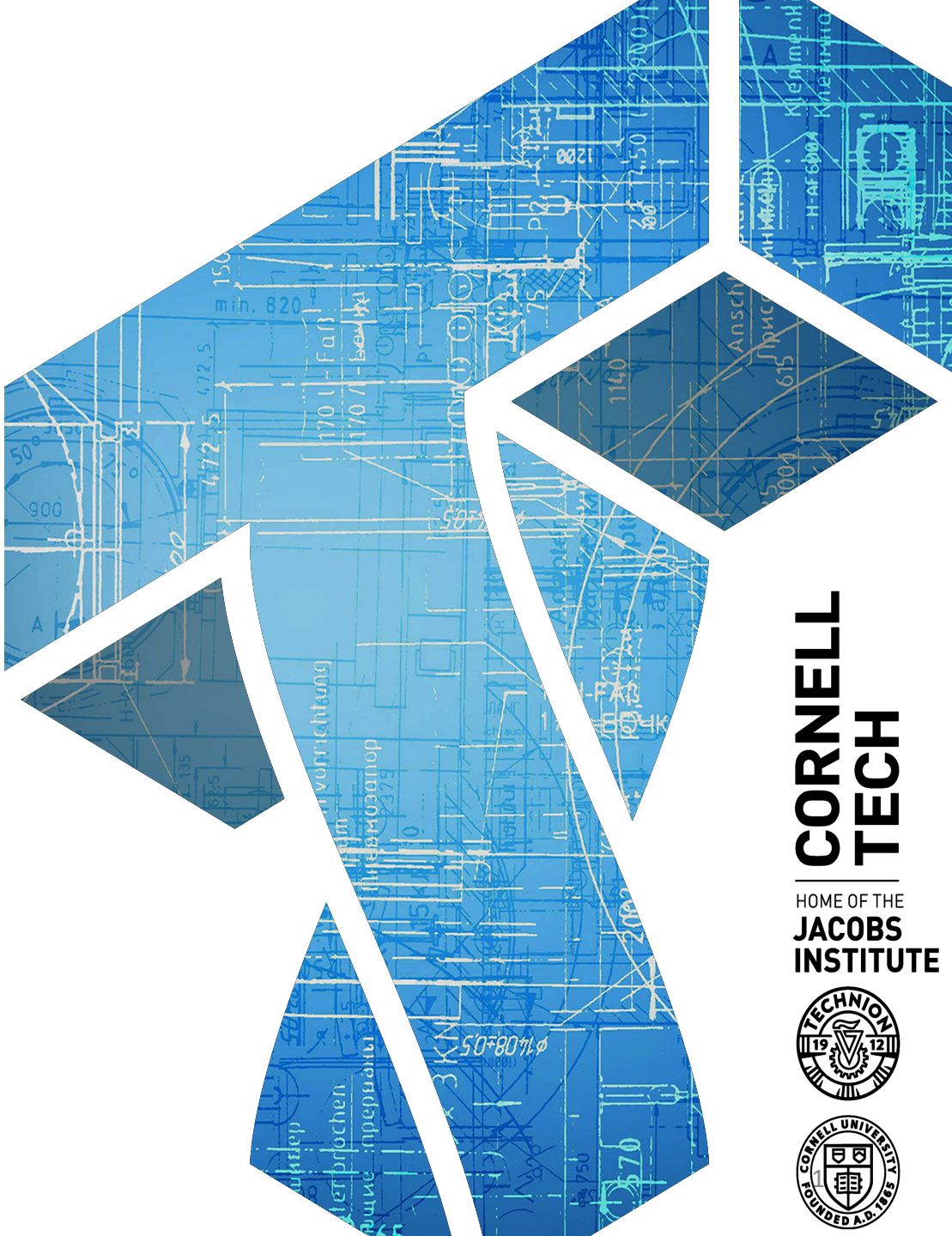
# CS 6431: Symmetric encryption & padding oracles

Instructor: Tom Ristenpart

https://github.com/tomrist/cs6431-fall2021



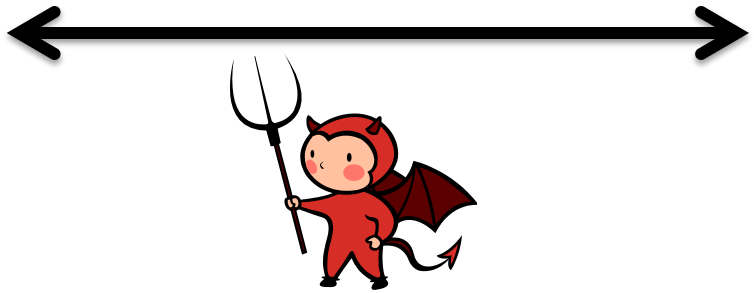
CORNELL TECH
HOME OF THE JACOBS INSTITUTE

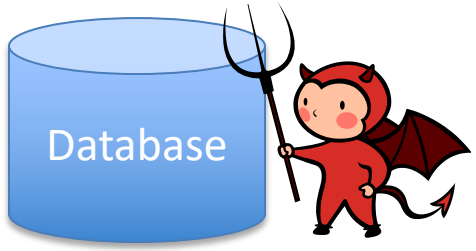
# Previously: outsourced encryption

- Client-side encryption for outsourced data
  - Tension of server-side functionality versus confidentiality
  - Property revealing encryption, structured encryption
- Today:
  - Backing up to standard symmetric encryption and understanding landscape and usage contexts
  - Particularly focus on chosen-ciphertext attacks

# Network adversary threat models

Client
(e.g., laptop,
workstation,
web server)






| Person | Age | Income |
|---|---|---|
| Alice | 22 | 100,000 |
| Bob | 29 | 25,000 |
| Charlie | 31 | 40,000 |

Server

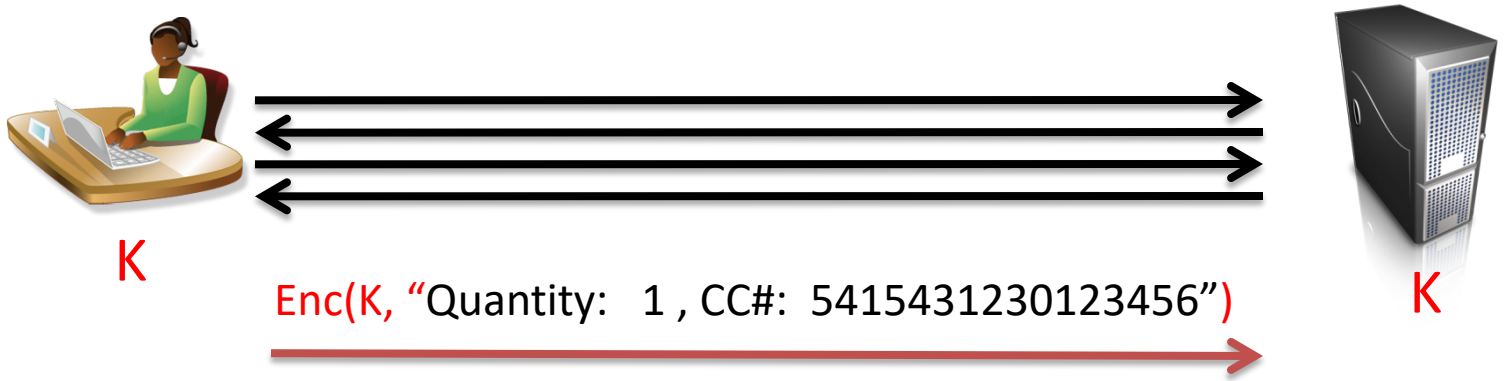
Network adversary is common threat model:

- On-path (sees packets) versus in-path (handles routing packets, can modify)
- Passive versus active

Cryptography helps us secure communications from network adversaries

# Transport Layer Security (TLS)

https://amazon.com




Step 1:
Key exchange protocol to share secret K

Step 2:
Send data via encrypted channel

TLS uses many cryptographic primitives:

**key exchange:** hash functions, digital signatures, public key encryption
**record layer:** symmetric encryption, message authentication

Security goals: confidentiality, integrity, server (and/or client) authenticity

Assumes strong adversaries that control network (observe, inject, reorder, drop, etc. packets)

# A short history of TLS

How many cryptographers involved?

SSL ver 2.0 designed by **Hickman** at Netscape

SSL ver 2

1994

**Wagner, Goldberg** break SSL ver 2

1995

**Freier, Karlton, Kocher** design SSL ver 3.0

SSL ver 3

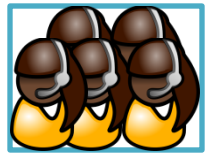
1998

**Bleichenbacher** breaks RSA PKCS #1 encryption, used in SSL ver 3

1999

**TLS ver 1** released as IETF standard, based on SSL 3, **many cryptographers** involved

TLS ver 1.0



2001

**Vaudenay, Klima et al.** padding attacks

2002

**Rogaway** IV re-use insecurity

2003

**Brumley, Boneh** remote timing attacks

2006

**TLS ver 1.1** released as standard

TLS ver 1.1

⋮

(more attacks and fixes)

# TLS focus of (case study for) large body of research

- **New kinds of attacks**
  - Padding oracle attacks (against RSA and CBC mode), traffic analysis, against protocol and/or implementations
- **Handshake design**
  - Forward secrecy, speeding up round trips (zero RTT handshake), post-quantum cryptography
- **Implementation improvements**
- **Formal analyses**
  - Security models, “pen-and-paper” proofs, (semi-)automated analysis

# TLS 1.2 handshake for RSA transport

Client

Server

Pick random Nc

ClientHello, MaxVer, Nc, Ciphers/CompMethods →

Pick random Ns

← ServerHello, Ver, Ns, SessionID, Cipher/CompMethod

Check CERT using CA public verification key

← CERT = ($pk_s$, signature over it by CA)

Pick random PMS
C <- Enc(pk,PMS)

C →

PMS <- Dec(sk,C)

ChangeCipherSpec,
{ Finished, PRF(MS, “Client finished” || H(transcript)) } →

← ChangeCipherSpec,
{ Finished, PRF(MS, “Server finished” || H(transcript’)) }

Bracket notation means contents encrypted

MS <- PRF(PMS, “master secret” || Nc || Ns )

# TLS handshake for Diffie-Hellman Key Exchange

Client

Server

Pick random Nc

ClientHello, MaxVer, Nc, Ciphers/CompMethods →

Pick random Ns

Pick random x

$X = g^x$

← ServerHello, Ver, Ns, SessionID, Cipher/CompMethod

Check CERT using CA public verification key

← $CERT = (pk_s$ , signature over it by CA)

Check σ

← $g , X , \sigma = Sign(sk_s, g || X)$

Pick random y

$Y = g^y$

$Y$ →

$PMS = g^{xy}$

$PMS = g^{xy}$

ChangeCipherSpec,

{ Finished, PRF(MS, "Client finished" || H(transcript)) } →

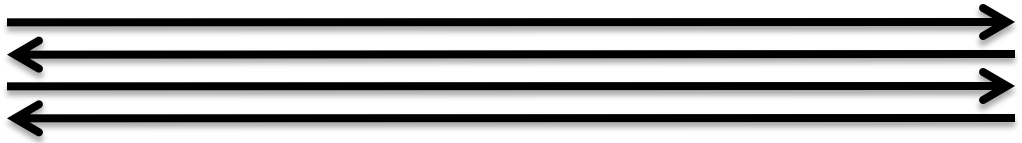
ChangeCipherSpec,

← { Finished, PRF(MS, "Server finished" || H(transcript')) }

Bracket notation means contents encrypted

MS <- PRF(PMS, "master secret" || Nc || Ns )

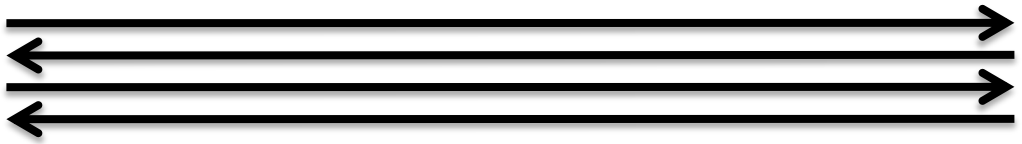
# Forward secrecy



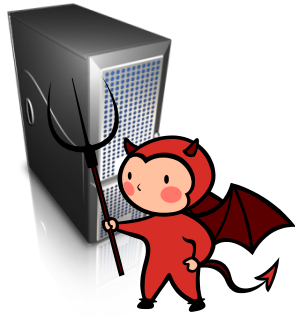
Enc(K, “Quantity: 1 , CC#: 5415431230123456”)

Record encrypted transcript

Can adversary recover previous plaintext data?



Enc(K, “this definitely leaks”)



Recover all long-lived secret keys

# Design of TLS 1.3 (latest version)

- TLS 1.3 was designed in a collaborative process
  - IETF was in charge
  - Eric Rescorla (Mozilla) coordinated effort
  - RFC 8446 (https://datatracker.ietf.org/doc/html/rfc8446)
- New designs discussed in public forums
- Combination of manual and automated analyses by academics
- Multi-year effort by dozens of people
- Forward secrecy only key exchange, zero RTT handshake mode, authenticated encryption with associated data (AEAD) for record layer, …

# Types of attacks against TLS

- Bad randomness [Wagner, & Goldberg 1996, Ristenpart & Yilek 2010, P's and Q's paper]
- CERT checking failures [Apple Double Goto, Georgiev et al. 2012, Fahl et al. 2012]
- Downgrade attacks [POODLE, DROWN]
- Bleichanbacher attacks [Bleichanbacher 1996]
- Backdooring TLS [Checkoway et al. 2014, 2016]

Key exchange

- Padding oracle attacks [Vaudenay 2002, Duong&Rizzo 2011, …]
- CBC mode predictable IV [Rogaway 1995, BEAST]
- Compression side-channel [Kelsey 2002, CRIME, BREACH]
- Traffic analysis attacks [See Dyer et al. 2012]

Record layer

# Symmetric encryption: Game plan

- Block ciphers
- IND$ encryption from block ciphers
- Padding oracle attacks
- Modern viewpoint: all-in-one authenticated encryption
- Emerging viewpoint: committing AEAD

# Reductionist approach to cryptography

Formal definitions

Scheme semantics

Security

Security proofs (reductions)

Breaking scheme

Breaking assumptions

Example:

**Attacker can not recover credit card**

Can not break block cipher

As long as assumptions holds
we believe in security of scheme!

Provable security yields

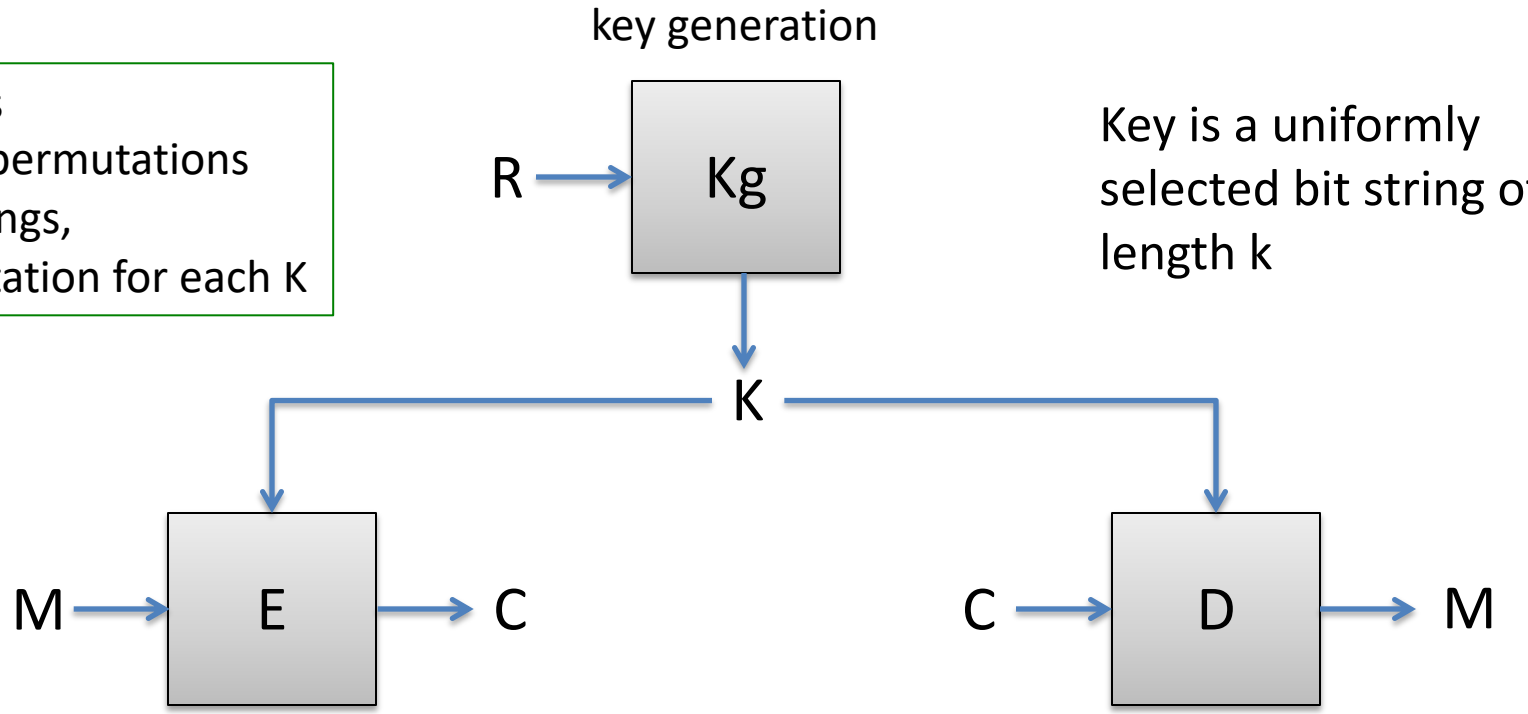
1) well-defined assumptions and security goals
2) cryptanalysts can focus on assumptions and models

But no one knows how to do this. It's been studied for a very long time!

# Block ciphers

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

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

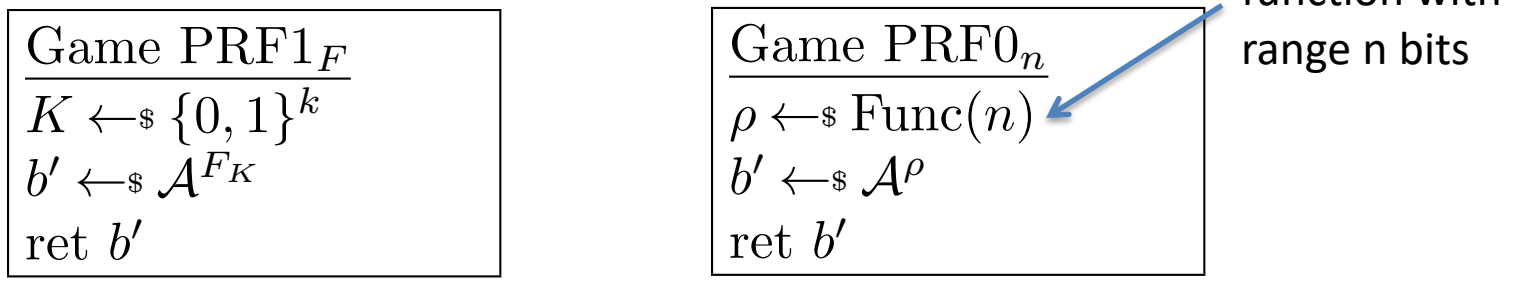



Key is a uniformly selected bit string of length k

Security goal: $E(K,\cdot)$ computationally indistinguishable from random function

# Pseudorandom functions (PRFs)

Can adversary distinguish between secret-keyed function and a random function?




$$\mathbf{Adv}_F^{\mathrm{prf}}(\mathcal{A}) = \left|\Pr\left[\,\mathrm{PRF1}_F^{\mathcal{A}} \Rightarrow 1\,\right] - \Pr\left[\,\mathrm{PRF0}_n^{\mathcal{A}} \Rightarrow 1\,\right]\right|$$

Measure adversarial resources concretely: running time and queries
Proving "low" advantage for large resources translated as providing security

Can also provide PRP notion (pseudorandom permutations). Same as above but with random permutation. Strong PRPs: give access to inverse as well

# Block cipher design

- Design of good block ciphers done by cryptanalyst community (visit Europe)
- Cryptanalysts have impressive laundry list of things to avoid when designing
  - Linear & differential cryptanalysis key techniques developed in 1990s

- We gain confidence by continued inability to break ciphers
- DES and AES competitions facilitated this

# Block cipher modes of operation

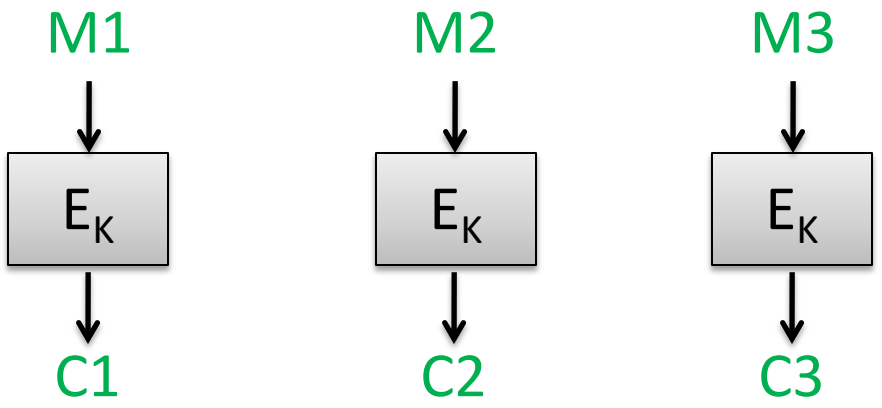
How can we build an encryption scheme for arbitrary message spaces using a block cipher?

- NIST modes from 1980s

Electronic codebook (ECB) mode

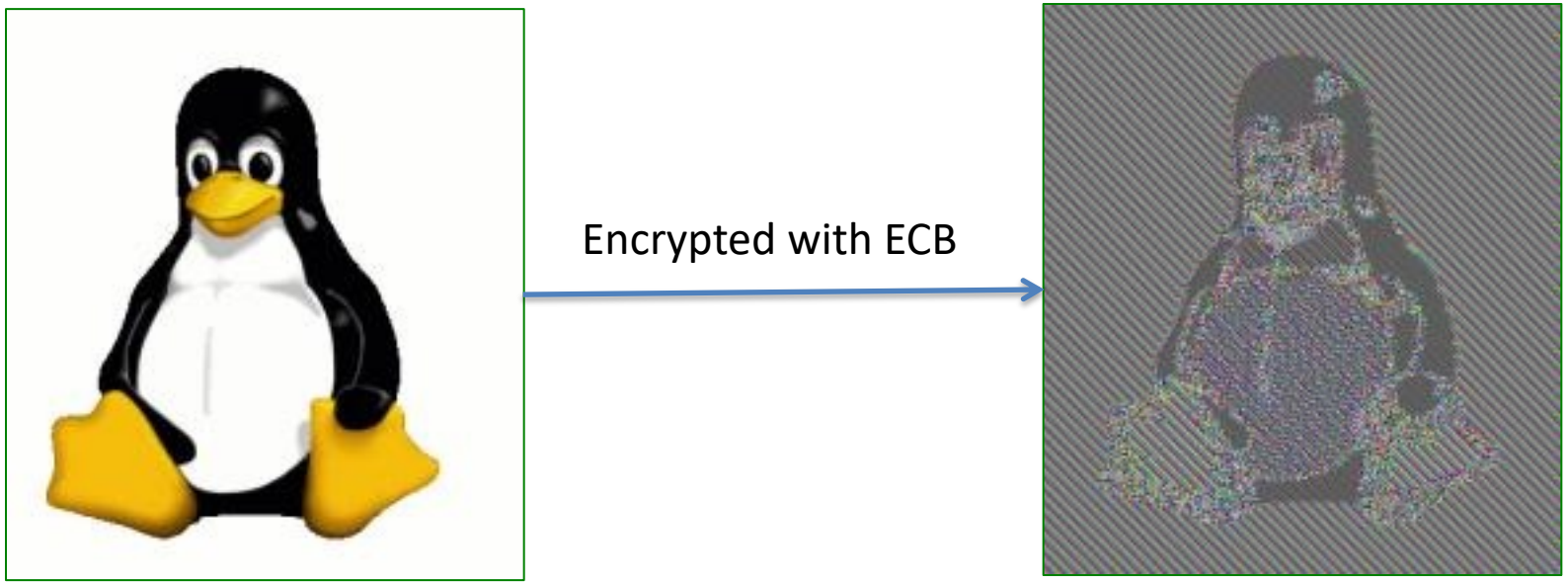
Pad message M to M1,M2,M3,... where each block Mi is n bits

Then:



# ECB mode does not provide confidentiality

ECB: substitution cipher with alphabet n-bit strings instead of digits




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

# Real-or-random indistinguishability (IND$)

Standard security goal for symmetric encryption

Can't distinguish encryption of chosen messages from random string of bits

$\underline{\mathrm{REAL}^{\mathcal{A}}_{\mathsf{SE}}}$
$K \leftarrow\!\!\$\ \mathsf{kg}$
$b' \leftarrow\!\!\$\ \mathcal{A}^{\mathrm{Enc}}$
Ret $b'$

$\underline{\mathrm{Enc}(M)}$
$C \leftarrow\!\!\$\ \mathsf{enc}_K(M)$
Ret $C$

$\underline{\mathrm{RAND}^{\mathcal{A}}_{\mathsf{SE}}}$
$b' \leftarrow\!\!\$\ \mathcal{A}^{\mathrm{Enc}}$
Ret $b'$

$\underline{\mathrm{Enc}(M)}$
$C \leftarrow\!\!\$\ \{0,1\}^{\mathsf{clen}(|M|)}$
Ret $C$

Adversary can repeatedly query Enc on same message

Adversary is a randomized algorithm with access to Enc oracle

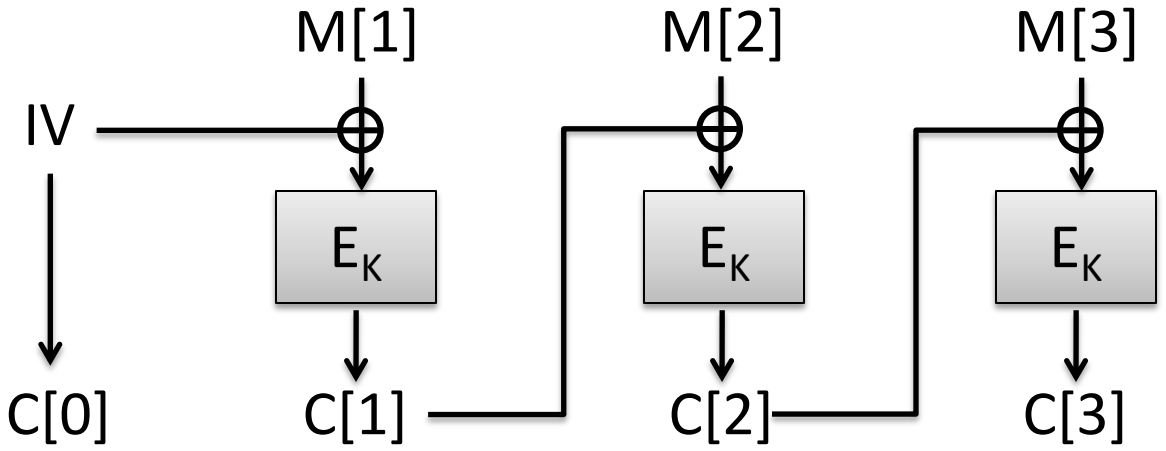
Advantage function measures efficacy of an adversary:

$$\mathbf{Adv}^{\mathrm{ind\$}}_{\mathsf{SE}}(\mathcal{A}) = \left|\Pr\left[\,\mathrm{REAL}^{\mathcal{A}}_{\mathsf{SE}} \Rightarrow 1\,\right] - \Pr\left[\,\mathrm{RAND}^{\mathcal{A}}_{\mathsf{SE}} \Rightarrow 1\,\right]\right|$$

Probabilities are over random coins used in games

# 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 SE scheme widely used

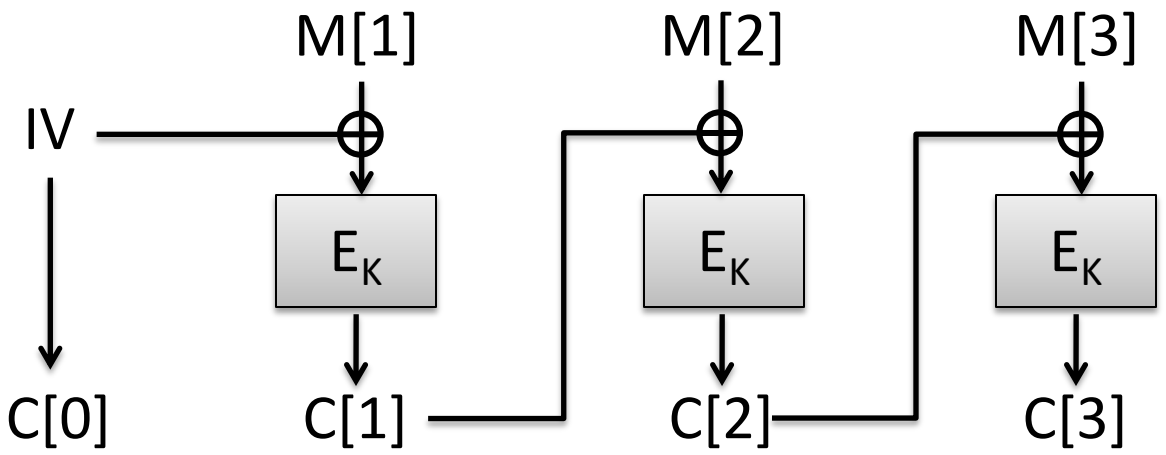
Kg():
K <-$ $\{0,1\}^k$

Pick a random key for blockcipher

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

Assume |M| is multiple of n
We'll see padding later on

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




Theorem (informal).
Let A be a successful, efficient IND$ attacker against security of CBC mode. Then we can construct a PRF adversary B against E that is efficient and successful.

Security proofs (reductions)

Breaking scheme

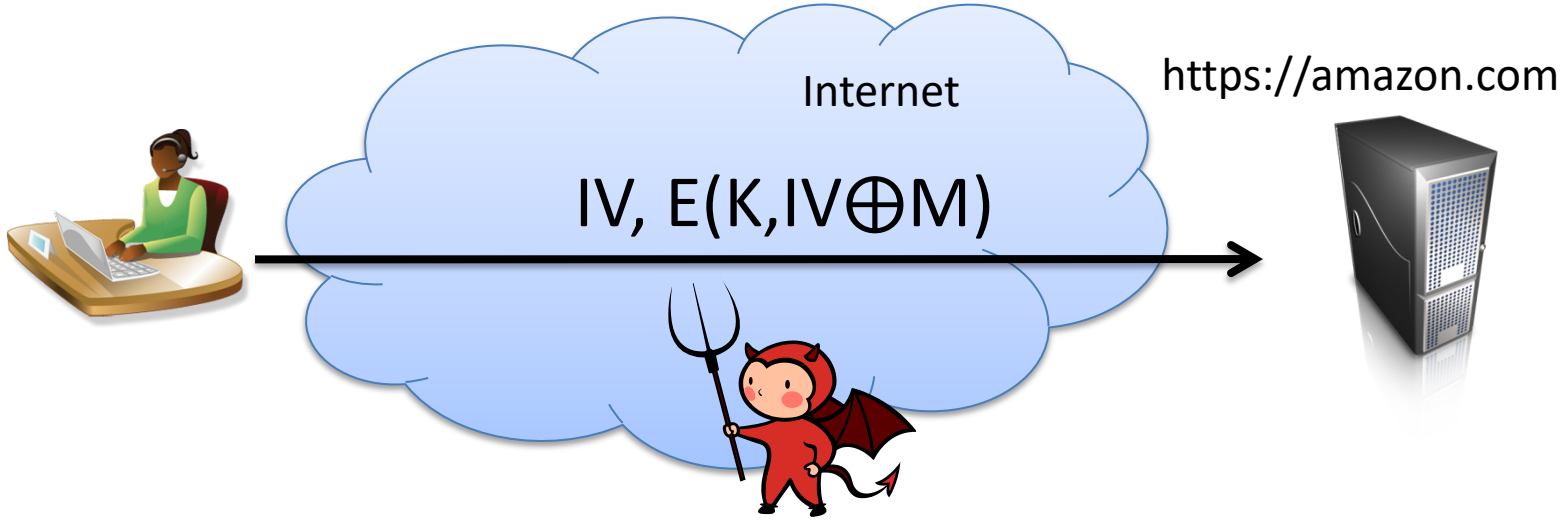
Breaking assumptions

Attacker can not break CBC confidentiality

Can not break E PRF sense

Reduces analysis now to E and to security definition / model

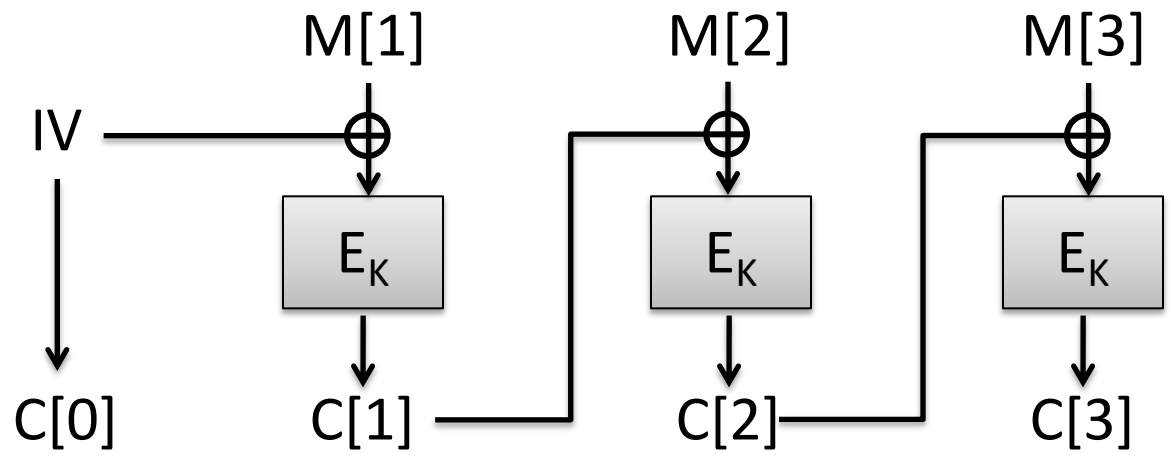
# Are we done?




We have shown IND$ confidentiality holds if blockcipher E is secure

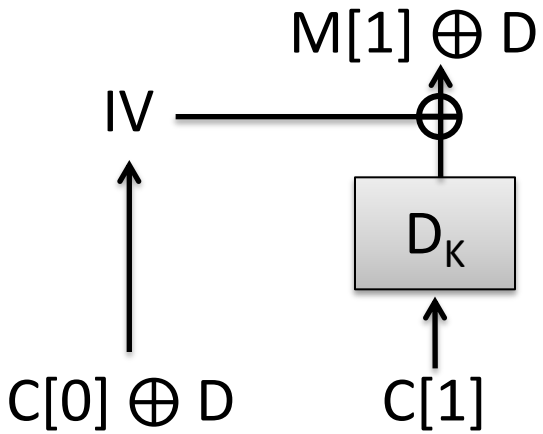
But we must also worry about active attackers that manipulate ciphertexts!

# CBC mode is malleable

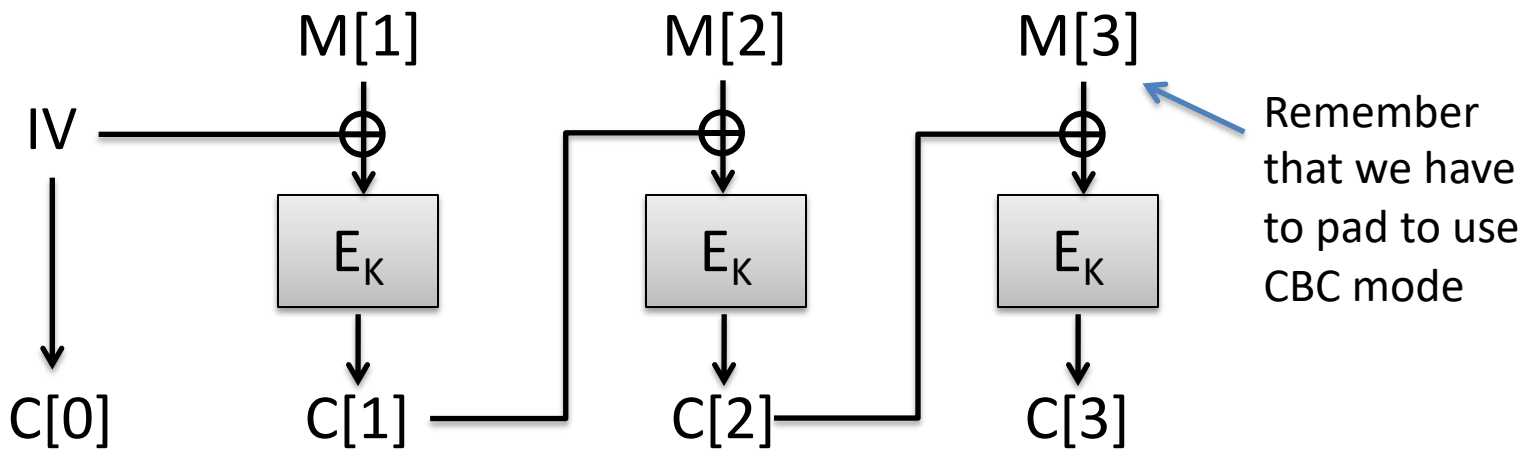



How do we change bits of M received by server in controlled way?

For any D:



# Padding oracle attacks




Padding oracle attacks take advantage of decryption implementations that reveal when padding checks fail

In most cases this allows *complete recovery of plaintexts*, violating confidentiality

Widespread damage: TLS, ASP.net, XML encryption, …

# 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 $C_1$

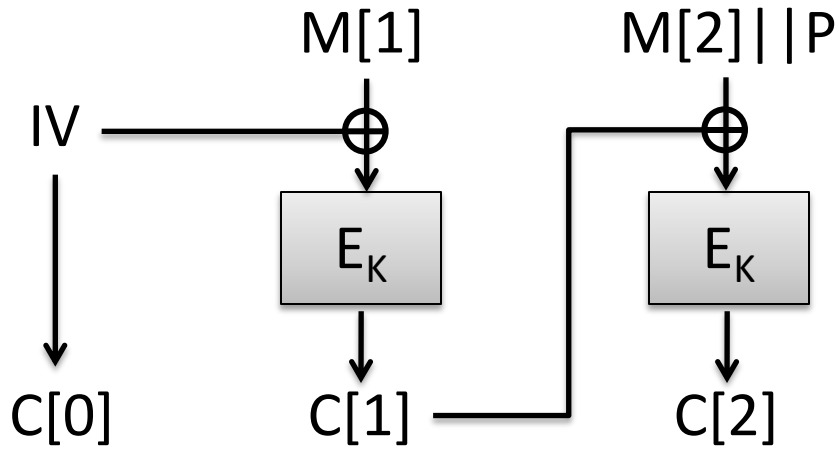
$C_4$ = Enc(K, “error”)

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]

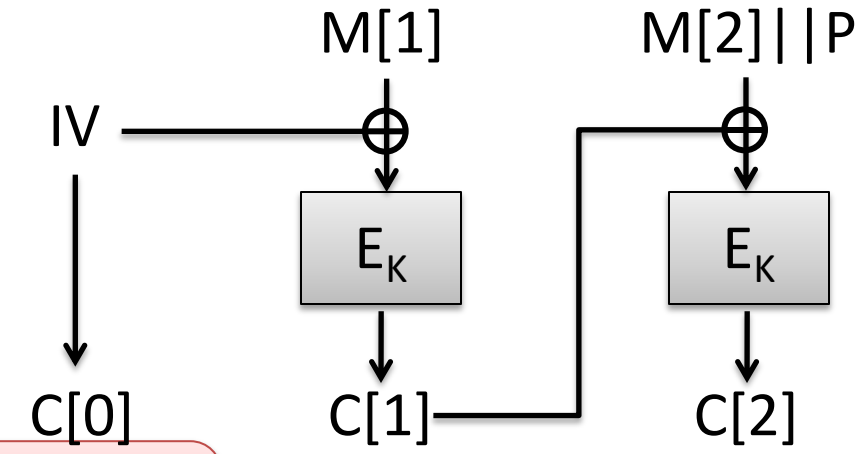
ok

C[0], C[1]⊕1 , C[2]

error

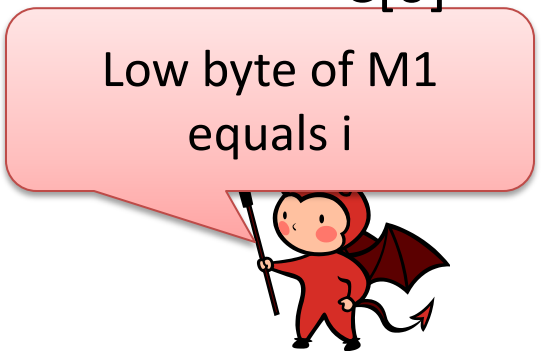
Dec(K, C' )
M'[1]||M'[2]||P' = CBC-Dec(K,C')
If P' ≠ 0x00 then
    Return error
Else
    Return ok

# 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




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

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

# PKCS #7 padding used most often with CBC mode

PKCS#7-Pad(M) = M || P || … || P

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])
For ctr < int(P):
        P’ = RemoveLastByte(M[m])
        If P’ != P then
                Return error
        ctr++
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]

C[0] , C[1]

error

C[0]⊕1 , C[1]

error

C[0]⊕2 , C[1]

error

…

C[0]⊕i , C[1]

ok

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

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

C[0][16] ⊕ X[1][16] = M[1][16]

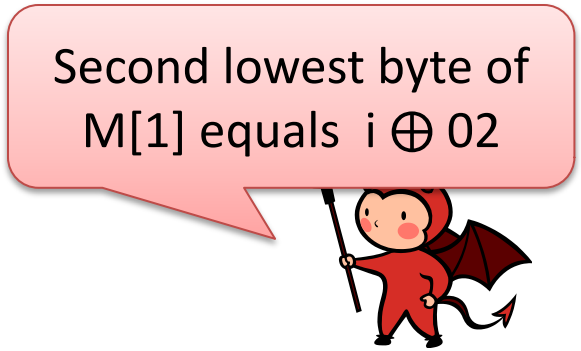
C[0][16] ⊕ i ⊕ X[1][16] = 01

M[1][16] ⊕ i = 01

Actually, it could be that:
M[1][16] ⊕ i = 02

Implies that M[1][15] = 02
We can rule out with an additional query

# PKCS #7 padding oracles




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

C[0]⊕0||j , C[1]

error

C[0]⊕1||j , C[1]

error

C[0]⊕2||j , C[1]

error

…

C[0]⊕i||j , C[1]

ok

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

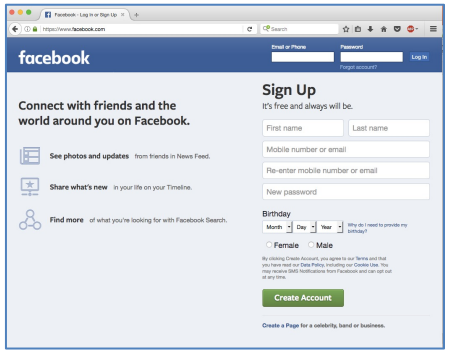
# Chosen ciphertext attacks against CBC

| 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 |

# Duong, Rizzo padding oracle work

- WOOT 2010 paper:
  - Using padding oracles to build ciphertext of arbitrary plaintext
  - Ways to find padding oracles in practice
  - Case study of problems
- Oakland 2011 paper:
  - Focus on ASP.net, a critical web framework by Microsoft

# Session handling and login




GET /index.html

Set-Cookie: AnonSessID=134fds1431

Protocol is HTTPS. Elsewhere else just HTTP.

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

Cookie: AnonSessID=134fds1431

Set-Cookie: SessID=83431Adf

Nowadays increasingly all HTTPS

GET /account.html

Cookie: SessID=83431Adf

# Security problems here?

website.com

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 = Enc(K, “admin=0”)

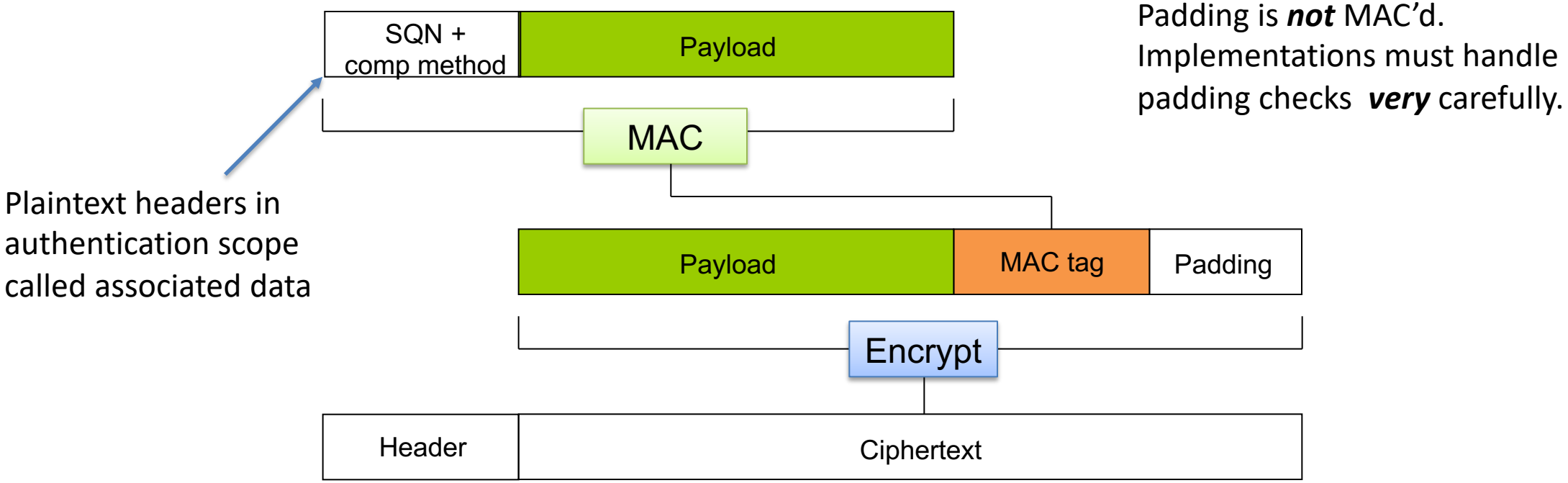
Malicious client can simply flip a few bits to change admin=1

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

# Authenticated encryption

- Malleability problems and chosen ciphertext attacks motivate ***authenticated*** encryption
  - No computationally bound adversary can forge a ciphertext that decrypts w/o error
- Actually authenticated encryption was already in place when Vaudenay's paper released

# TLS 1.2 record protocol: MAC-Encode-Encrypt (MEE)




The MAC (message authentication code) can be thought of as a pseudorandom function (PRF). During decryption:

Re-run MAC over header + Payload, check the given value

The hope is that this prevents manipulation of ciphertexts by adversaries, preventing padding oracle attacks

# (Simplified) Early TLS decryption

```
TLS-Decrypt(K, C )
M’[1] || … || M’[m] = CBC-Dec(K,C)
P = RemoveLastByte(M’[m])
while ctr < int(P):
        P’ = RemoveLastByte(M’[m])
        If plen’ != plen
                Send PADDING_ERROR message
        ctr++
(X,tag) <- ParseLast20bytes(M’[1] || … || M’[m])
If MAC(K,X) != tag then
        Send MAC_ERROR MESSAGE
Return X as plaintext to application
```

How do we mount an attack?

# Evolution of views on symmetric encryption

- Encryption is for confidentiality against passive adversaries
- Add explicit message authentication layer to provide plaintext integrity
- Need “all-in-one” authenticated encryption modes that simultaneously provide confidentiality and ciphertext integrity
  - Doesn’t say much if IV is chosen incorrectly, or if adversary has control over keys
- Need authenticated encryption that is as robust as possible to other threat models:
  - IV (nonce) reuse, side channel resistance, key committing, …