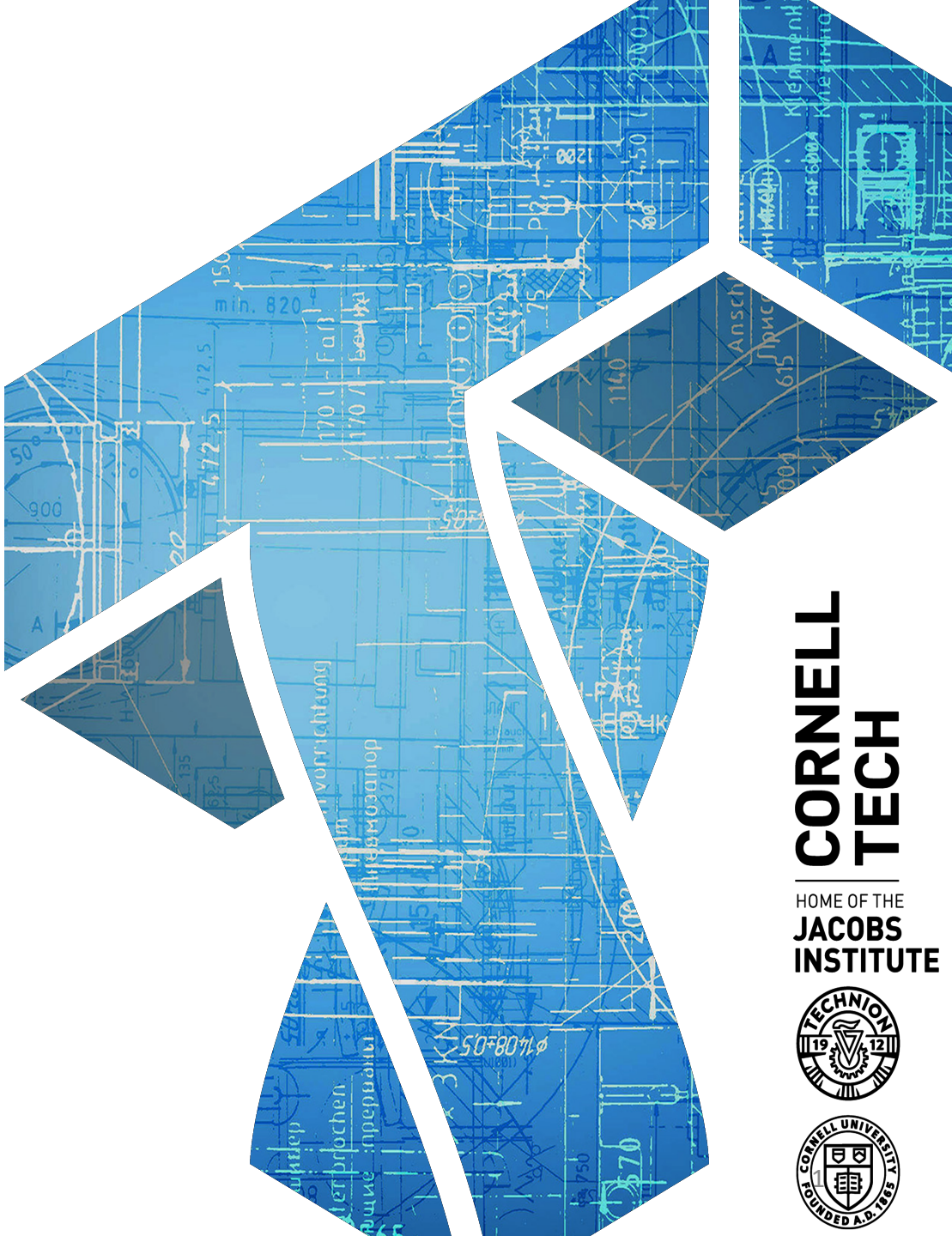
CS 6431: Padding oracles

Instructor: Tom Ristenpart

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

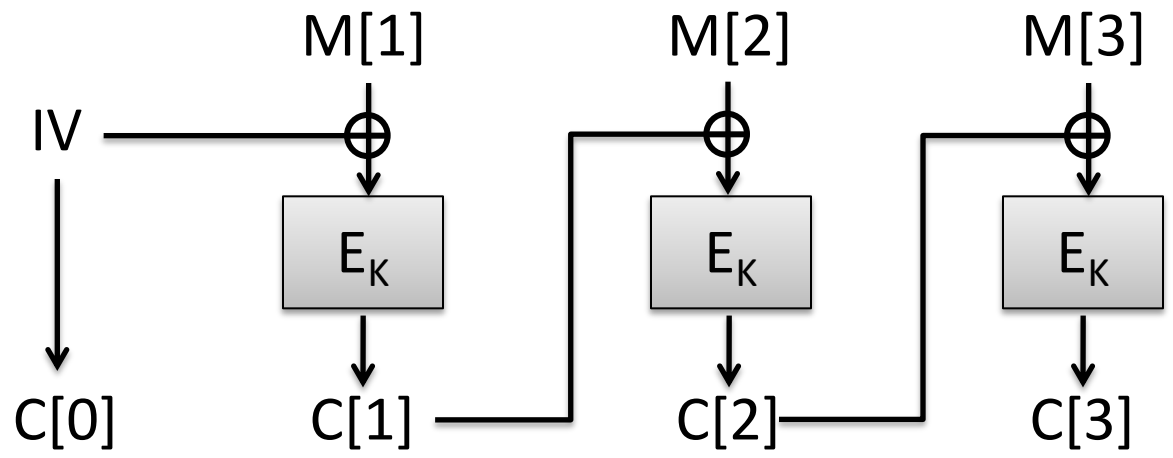


CORNELL TECH
HOME OF THE JACOBS INSTITUTE

Proposal & reviews being reviewed

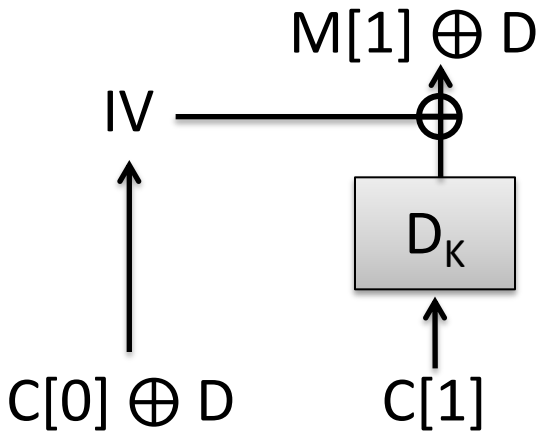
- So far nice job!
  - A few missing reviews this morning. You will be docked for late reviews, reach out to explain please.
- Feel free to discuss proposals/reviews on HotCRP
  - Get clarification from reviewers
  - Provide clarification to reviewers
- I hope to finish going over all the project proposals by the end of the weekend. Will upload reviews
- Questions?

CBC mode is malleable

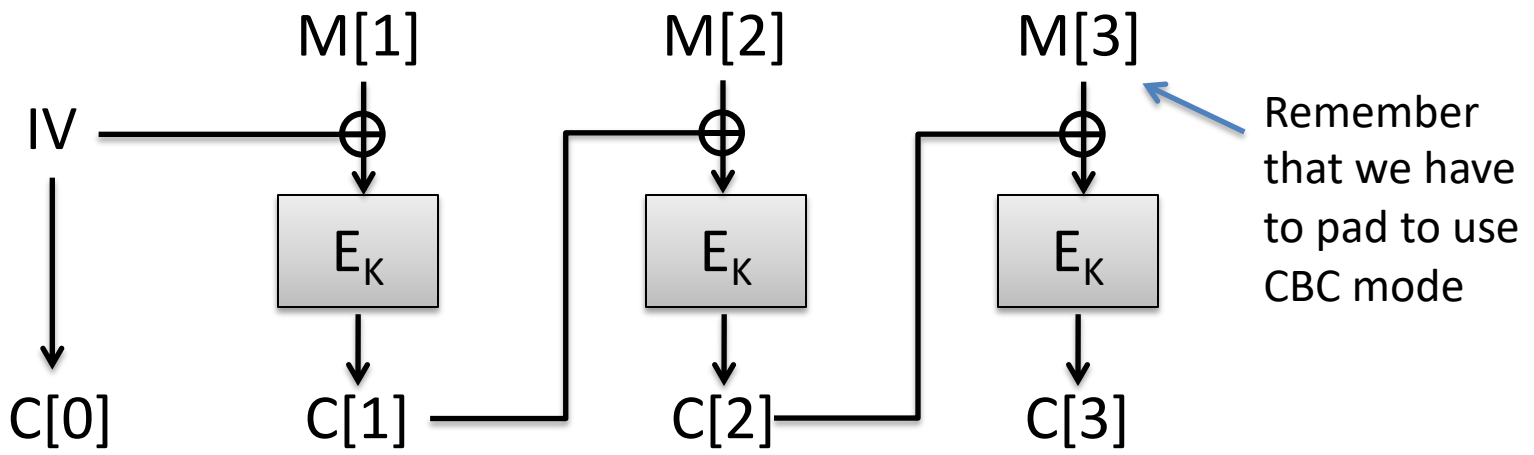


How do we change bits of M received by server in controlled way?

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 oracle threat model

- Attacker obtains encrypted message(s) enc(K,M) for some unknown M
- Attacker wants to recover (information about M)
  - Can query a ***partial decryption oracle*** that does not reveal M, but some limited information about decryption process

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

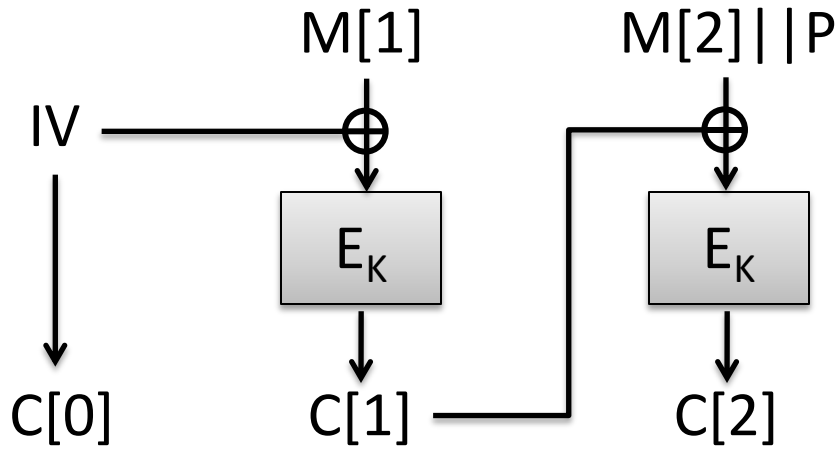
C_4 = Enc(K, “error”)

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]

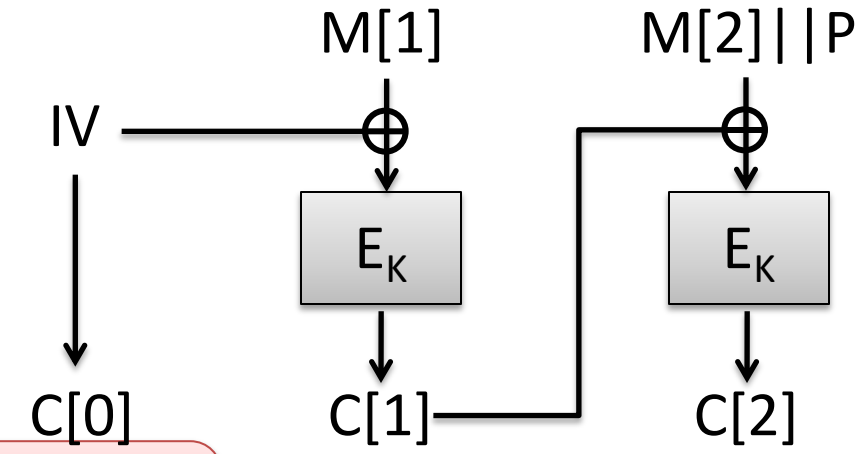
ok

C[0], C[1]⊕1 , C[2]

error

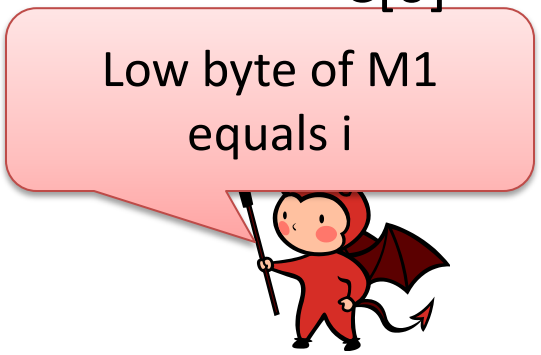
Dec(K, C')
M'[1]||M'[2]||P' = CBC-Dec(K,C')
If P' ≠ 0x00 then
    Return error
Else
    Return ok

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

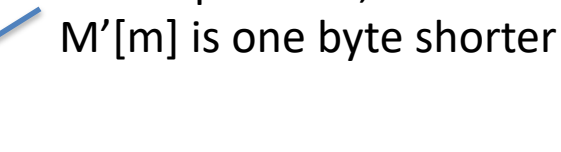
Possible paddings:

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

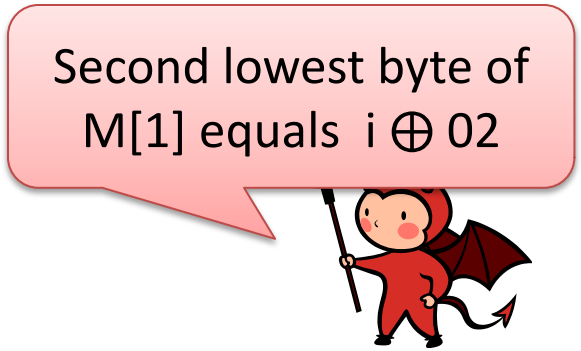
Actually, it could be that:

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

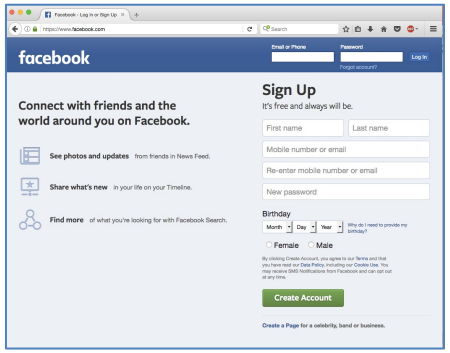
Chosen ciphertext attacks against CBC

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***

Duong-Rizzo show how to turn into padding oracle, do all sorts of damage

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
  - “The best attack requires less than 2,000 HTTP requests to steal secret cryptographic keys in ASP.NET applications. After obtaining those keys, the attacker can easily create authentication tickets that he can use to sign in to any user account.”

Security problems here?

POST /WebResource.axd?d=encrypted_id&t=timestamp

HTTP 500 or HTTP 404

ASP.net website

Secret key K only known to server

encrypted_id = CBC-Enc(K, R#anything || ~/path/to/file)

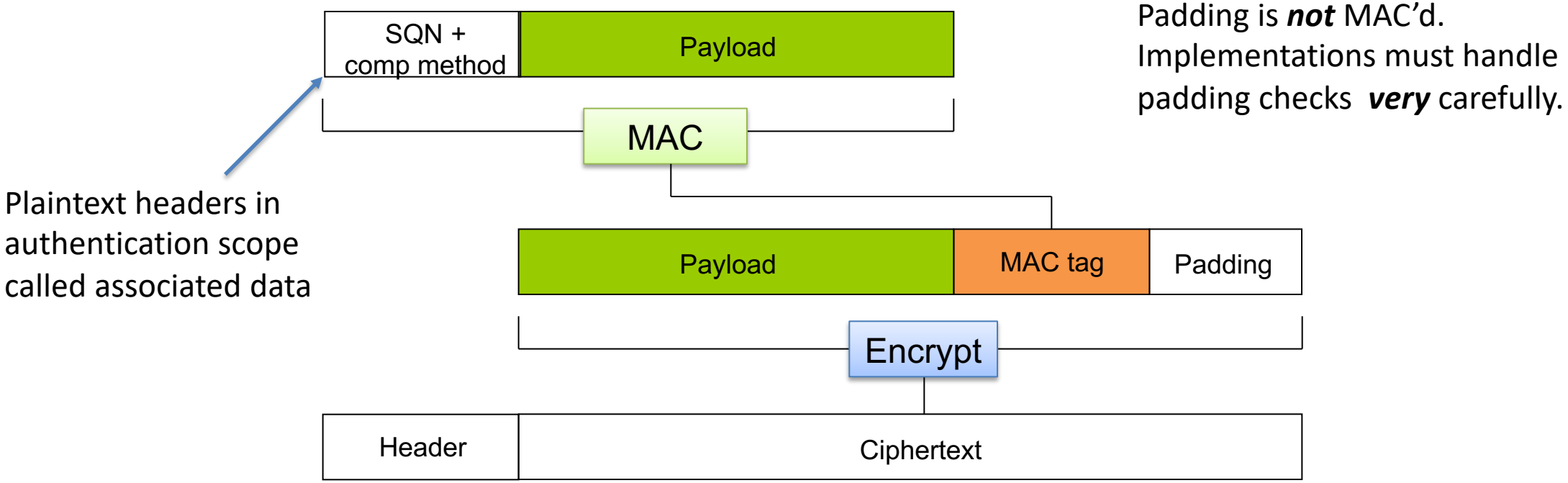
Algorithm 3 Downloading files using CBC-R and the padding oracle in `WebResource.axd`.

1) Pad the messsage `|||~/path/to/file` according to the PKCS#5 padding scheme, and set the whole result as the target P.
2) Divide P into n blocks of b bytes, denote them $P_1, P_2, \ldots, P_n$.
3) Use CBC-R to build $C_n, \ldots, C_2, C_1$ so that $C_1|C_2|\ldots|C_n$ decrypts to $P_{garbage}|P_1|P_2|\ldots|P_n$.
4) Pick a few random bytes $r_1, r_2, \ldots, r_b$, and set $C_0 = r_1|r_2|\ldots|r_b$.
5) Set $d = C_0|C_1|...|C_n$, and send it to `ScriptResource.axd` to see if `/path/to/file` is downloaded. If no, go back to step 4.
6) Output d.

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?

Authenticated encryption (w/ associated data)

- Encrypt-then-MAC generic composition
  - CTR mode encryption of message
  - HMAC or other MAC of ciphertext (including IV, associated data)
- Dedicated AEAD modes:
  - AES-GCM (Galois Counter Mode)
  - ChaCha2020/Poly1305
  - Offset Codebook mode (OCB)

Evolution of views on symmetric encryption

1. Encryption is for confidentiality against passive adversaries
2. Add explicit message authentication layer to provide plaintext integrity
3. 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
4. Message franking [Facebook, Grubbs et al. 2017], partitioning oracle attacks [Len et al. 2020] show partial decryption oracle attacks against AES-GCM, other widely used schemes
5. Need authenticated encryption that is as robust as possible to other threat models:
   – IV (nonce) reuse, side channel resistance, key committing, …