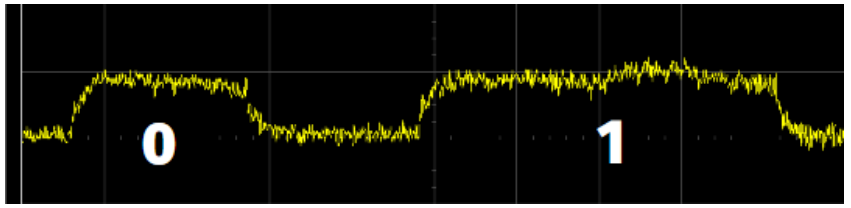
Lecture 11: Padding & padding oracles

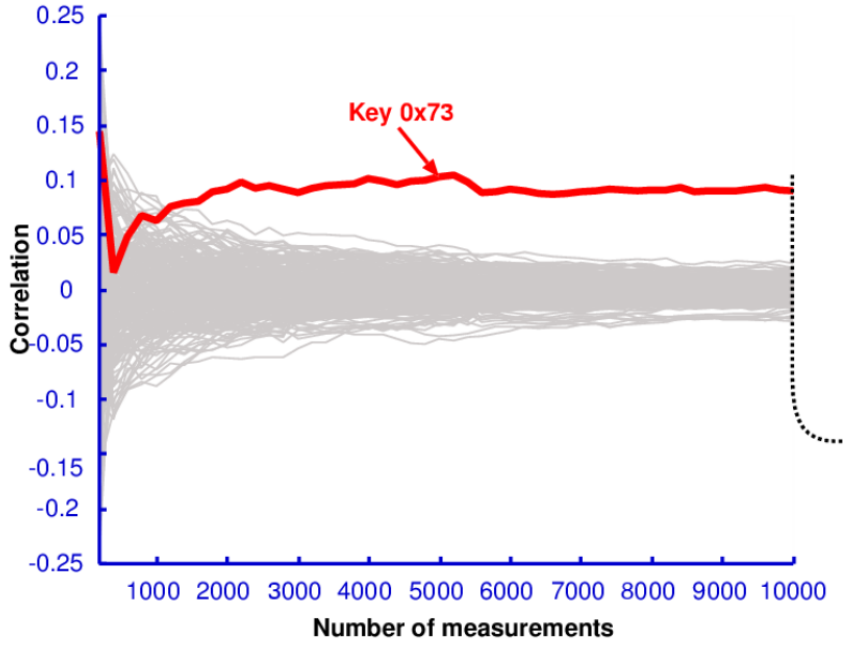
Announcements

- I am not Prof. Varia
- Lab 5 due Friday 3/8 at 11pm
- Prof. Varia will be back Thursday 3/7
- Office hours: Thursday 3/7 from 9-10 am and 2:30-3:30 pm
- Nicolas' office hours: normal time on Friday

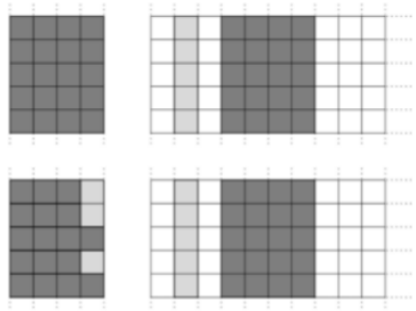
Last week: Power analysis and timing attacks

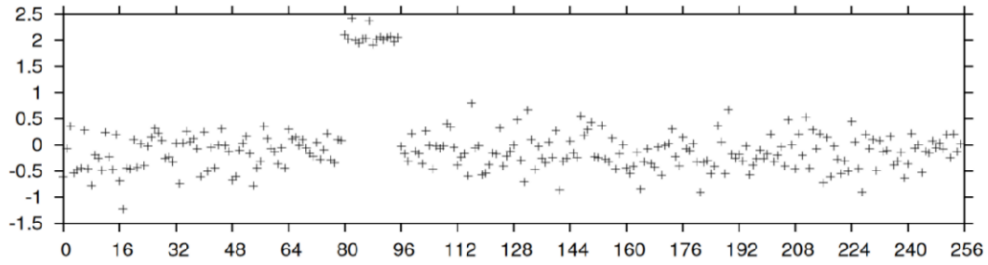
Power analysis (SPA, DPA, template)





Timing attacks (cache: prime+probe, evict+time; network)



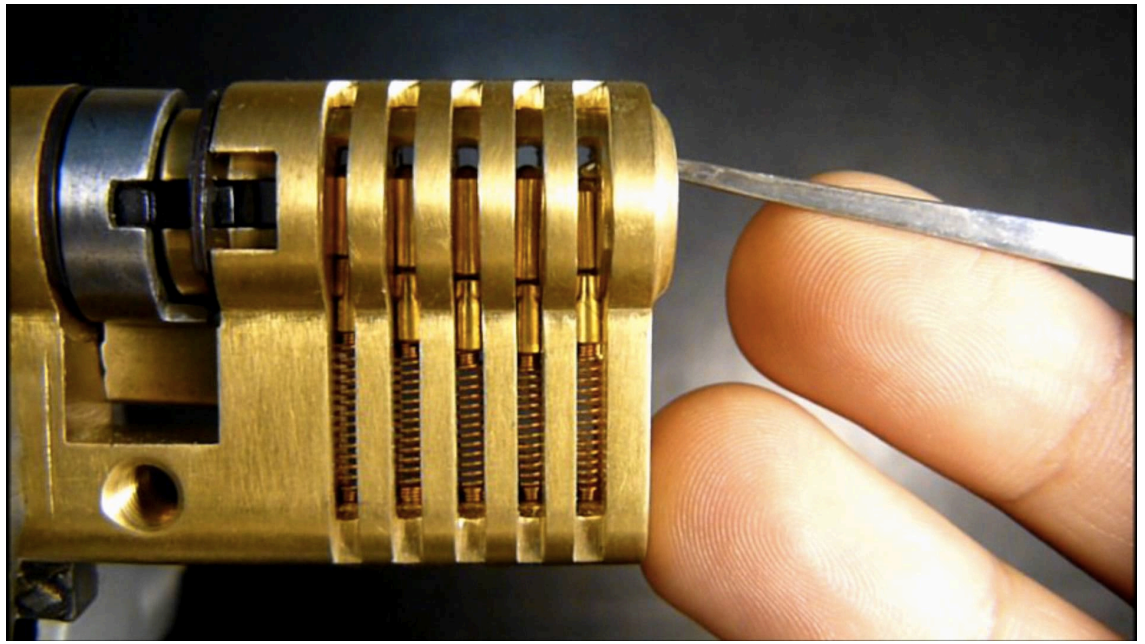


Today: Padding Oracle Attacks

- Last week: Attacks on *AES*
  - Exploit knowledge of power, timing
- Today: Attacks on *modes* of AES: CBC mode (plus other ingredients)
  - Exploit knowledge of error messages

Divide and conquer

- Attacks follow a *divide and conquer* approach: break 1 byte at a time
- For each byte, simply *guess* all 256 values *and check* which one works
- (Think: how you see crypto broken in any Hollywood movie)

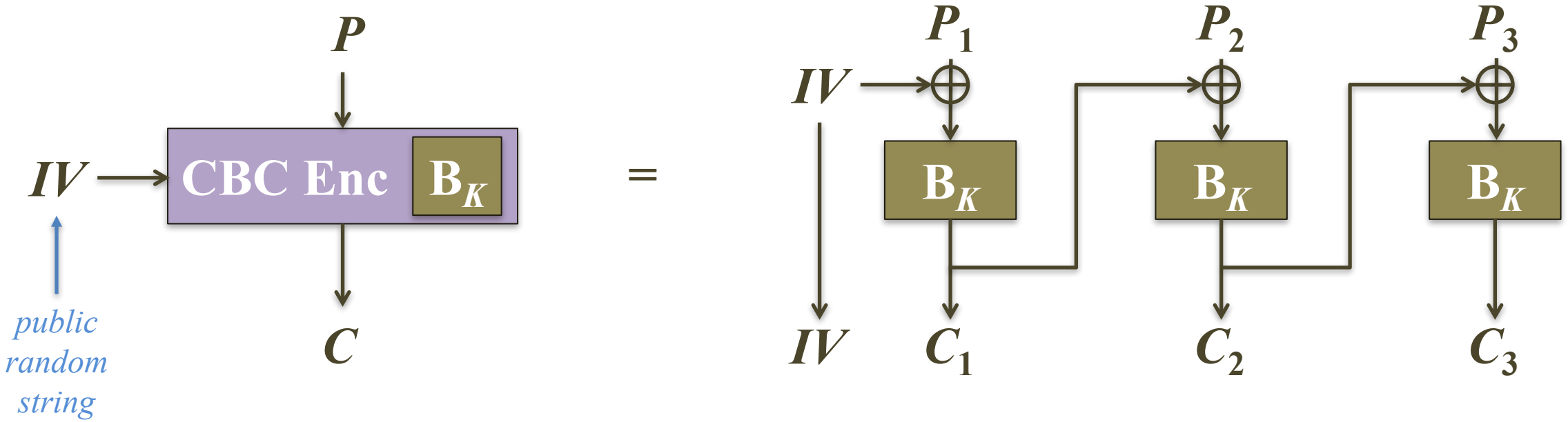


Padding Oracles [Vaudenay 2002]

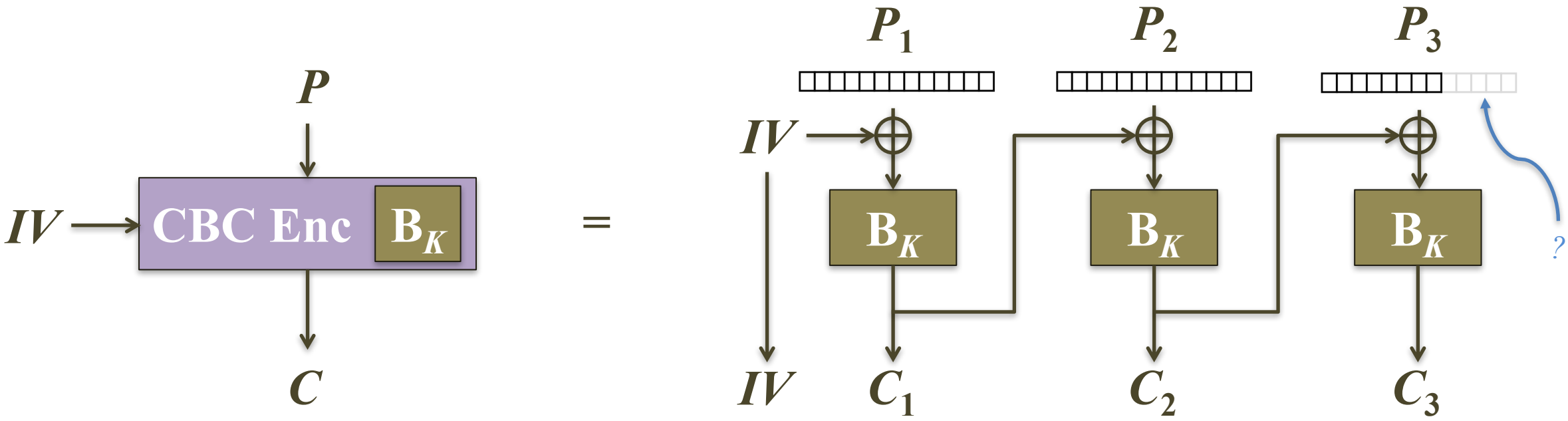
- Main idea: Exploit error messages for different kinds of malformed input to recover the plaintext

- Three building blocks:
  1. CBC Mode
  2. Error messages
  3. MAC-then-Encrypt

Building block 1: CBC mode (encryption)



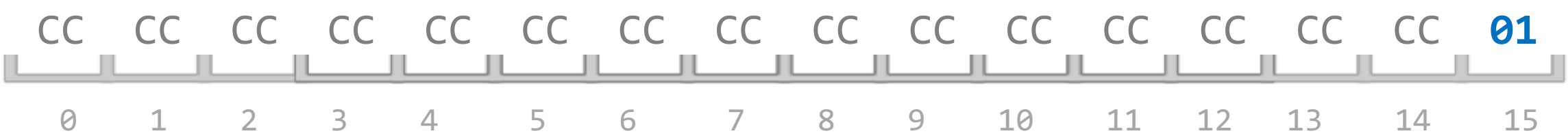
Recall: CBC mode needs **padding**

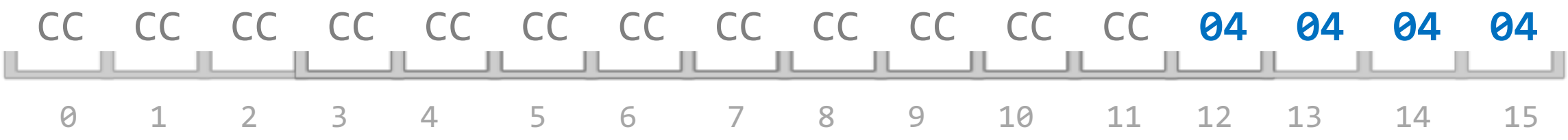


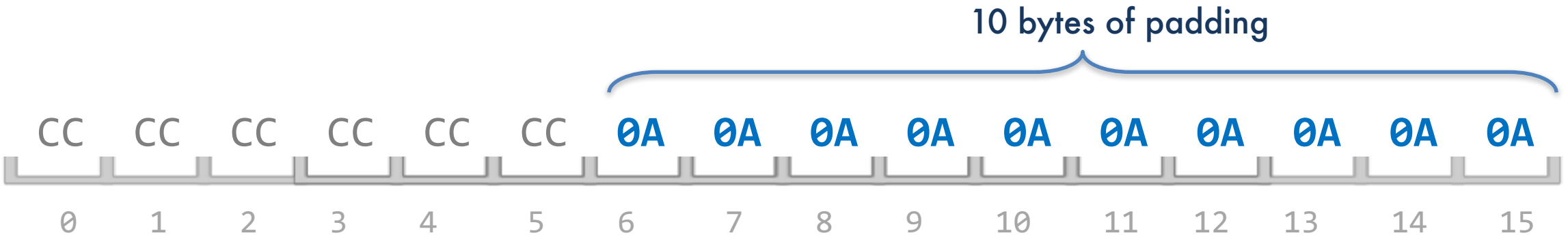
For today: Length of *P* = any number of bytes

- Will not "split" bytes
- Might not be multiple of 16

PKCS #7 padding

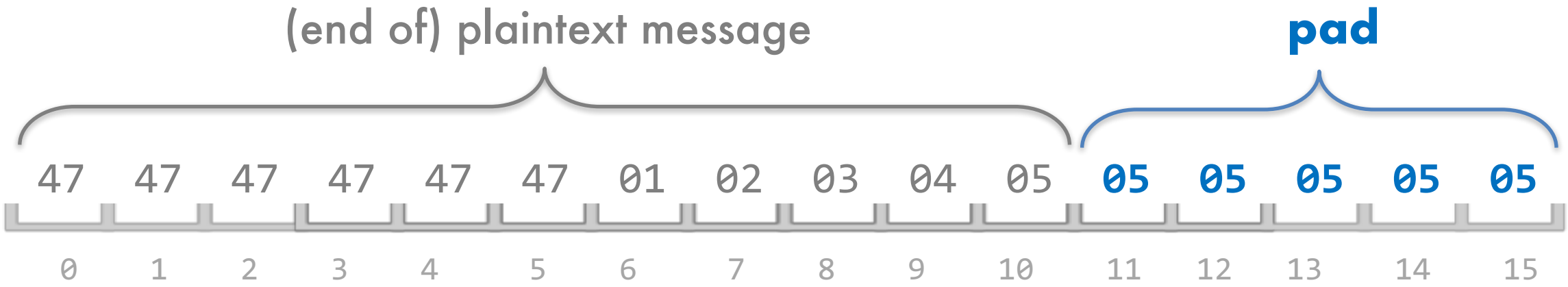




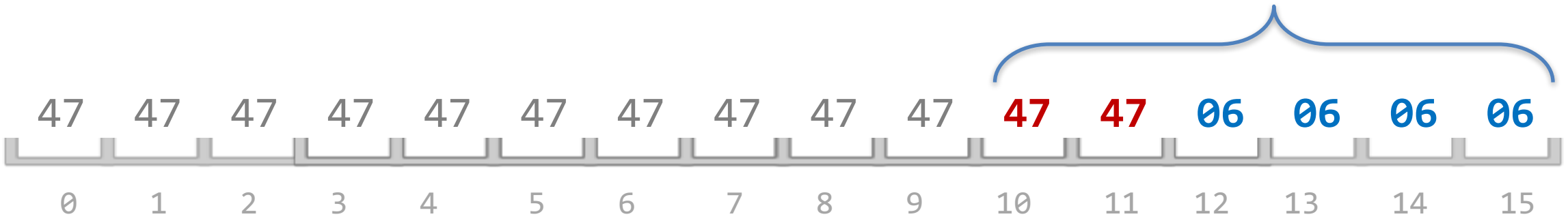


PKCS #7 padding

Padding adds N whole bytes, each of value N



What if padding is invalid?



Return error message "Invalid Padding"

(Building block 2 for padding oracle attacks)

Building block 3: MAC-then-Encrypt

So far in this class:

Privacy XOR Authenticity

Privacy: Encryption scheme
Authenticity: MAC

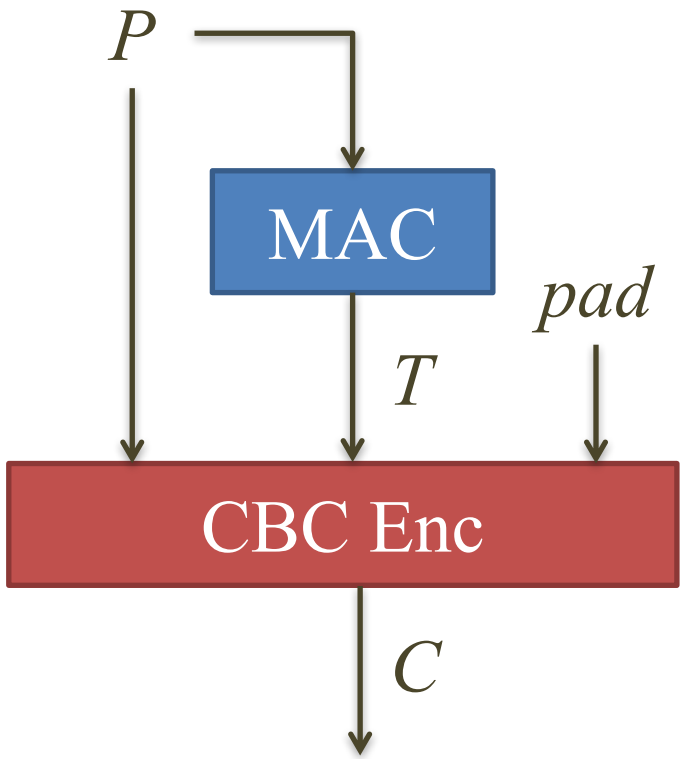
How do we put these together?

- Attempt 1: MAC-then-encrypt
- More on this later…

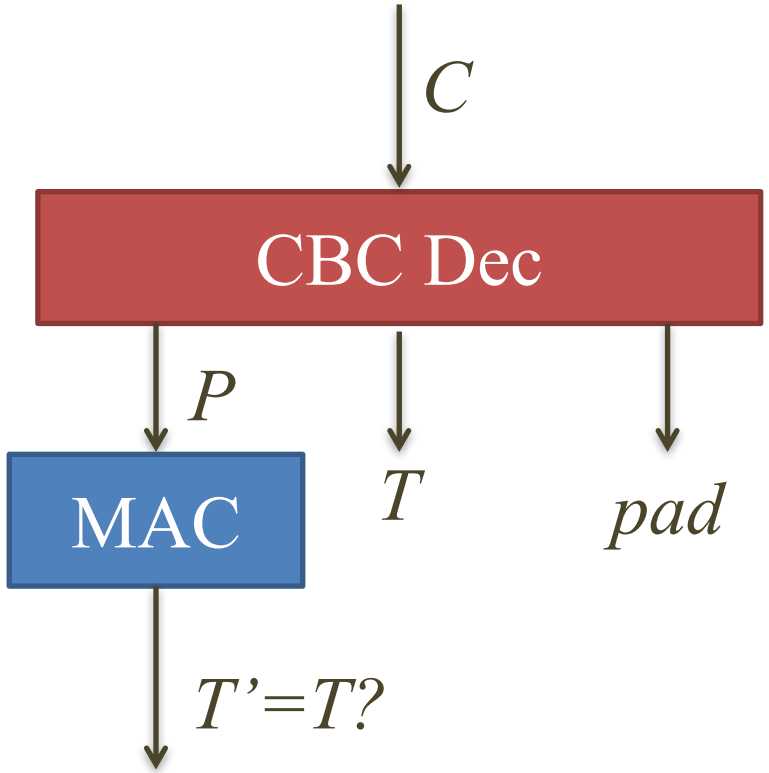
Building block 3: MAC-then-Encrypt

Consider the following scheme for encrypting and authenticating plaintext P:

1. Let T = HMAC(P)
2. Let pad = PKCS7(P || T) (|| is concatenation)
3. Return C = CBC_Enc(P || T || pad)



Decryption and Verification



Decrypting/authenticating ciphertext C:

1. Let (P, T, pad) = CBC_Dec(C)

Invalid Padding

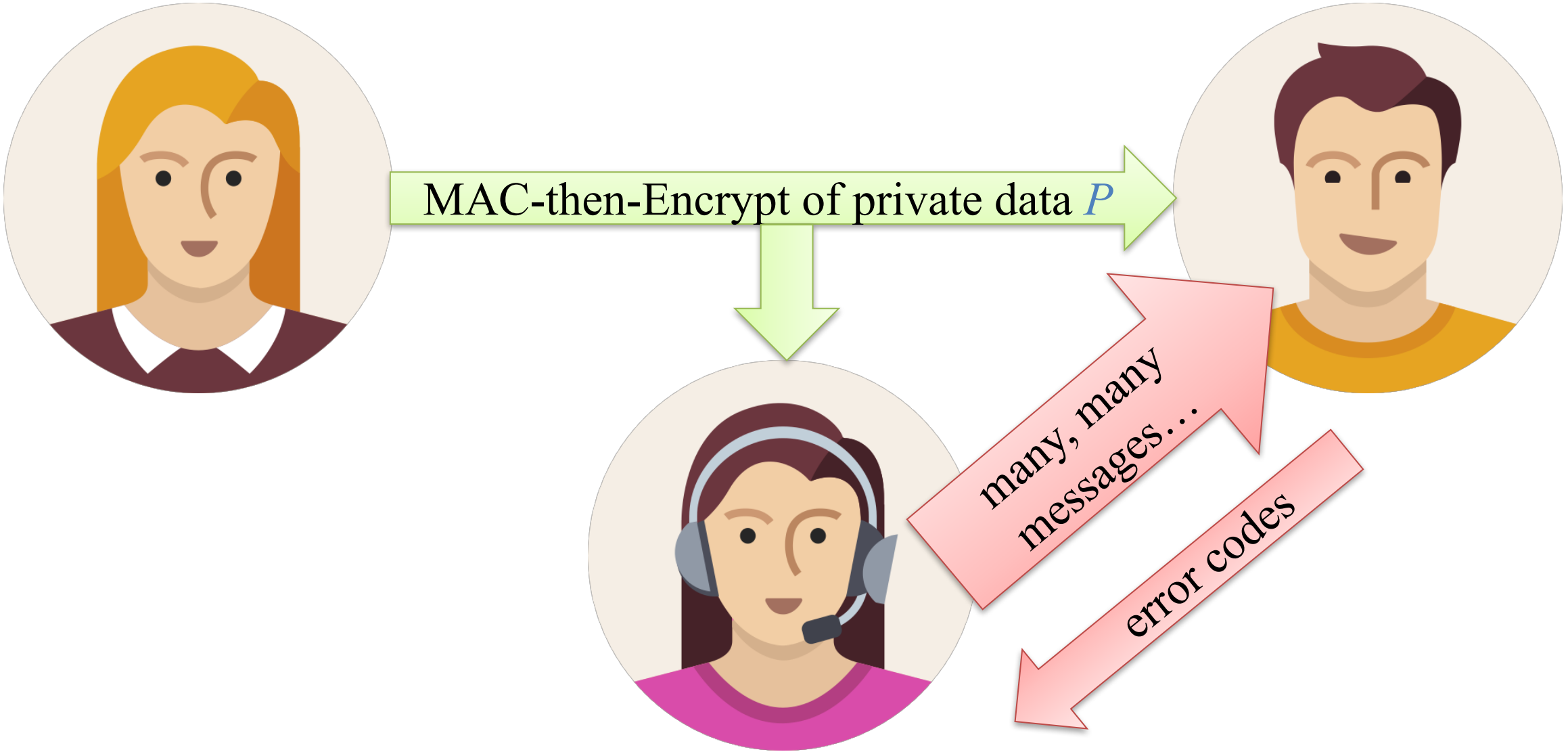
2. Let T′ = HMAC(P)
3. Check whether T′=T

Exploit these distinct error messages to recover P

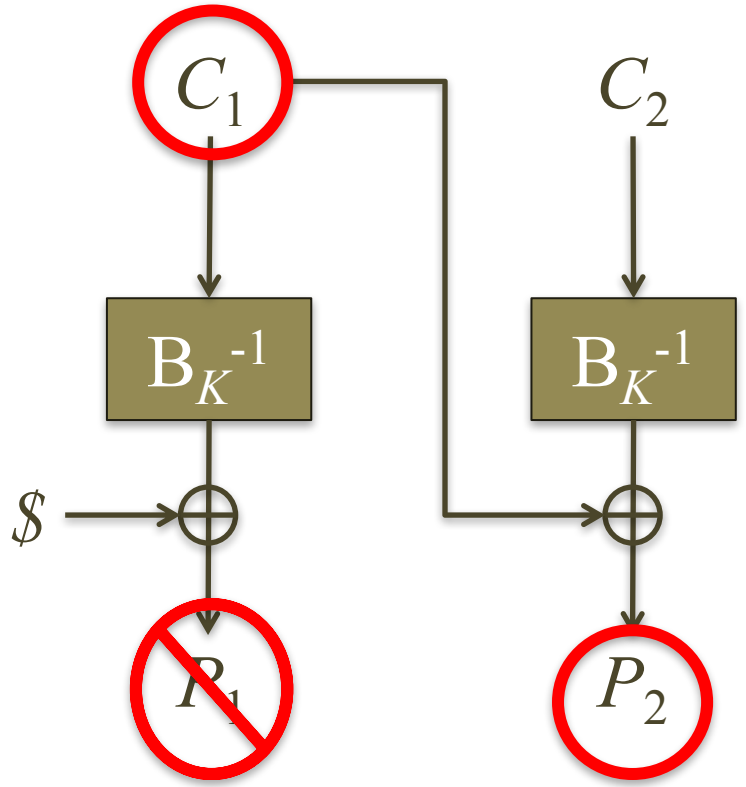
Invalid MAC

Valid plaintext

Attack setup



Problems with CBC decryption?



- Formally:
  - Doesn't provide integrity
  - Isn't nonce-respecting
  - …
- Specific concerns to exploit today:
  - Propagation only goes *forward*… so it suffices to design a mechanism that recovers the final block P_2
  - Malleability: altering ciphertext block C_1 changes plaintext block P_2 *in a byte by byte manner*! (Destroys P_1 in the process, but no matter)

Padding oracle attack: the idea

- Mallory knows C for unknown msg
- She uses mauling power to make all 256 options of final byte
- Exactly one will have the final byte 01 & thus look like a valid pad!
- The other mauled messages will have an invalid tag, such as 030302

private P	tag T	pad 030303

private P	tag T'	pad 0303xx

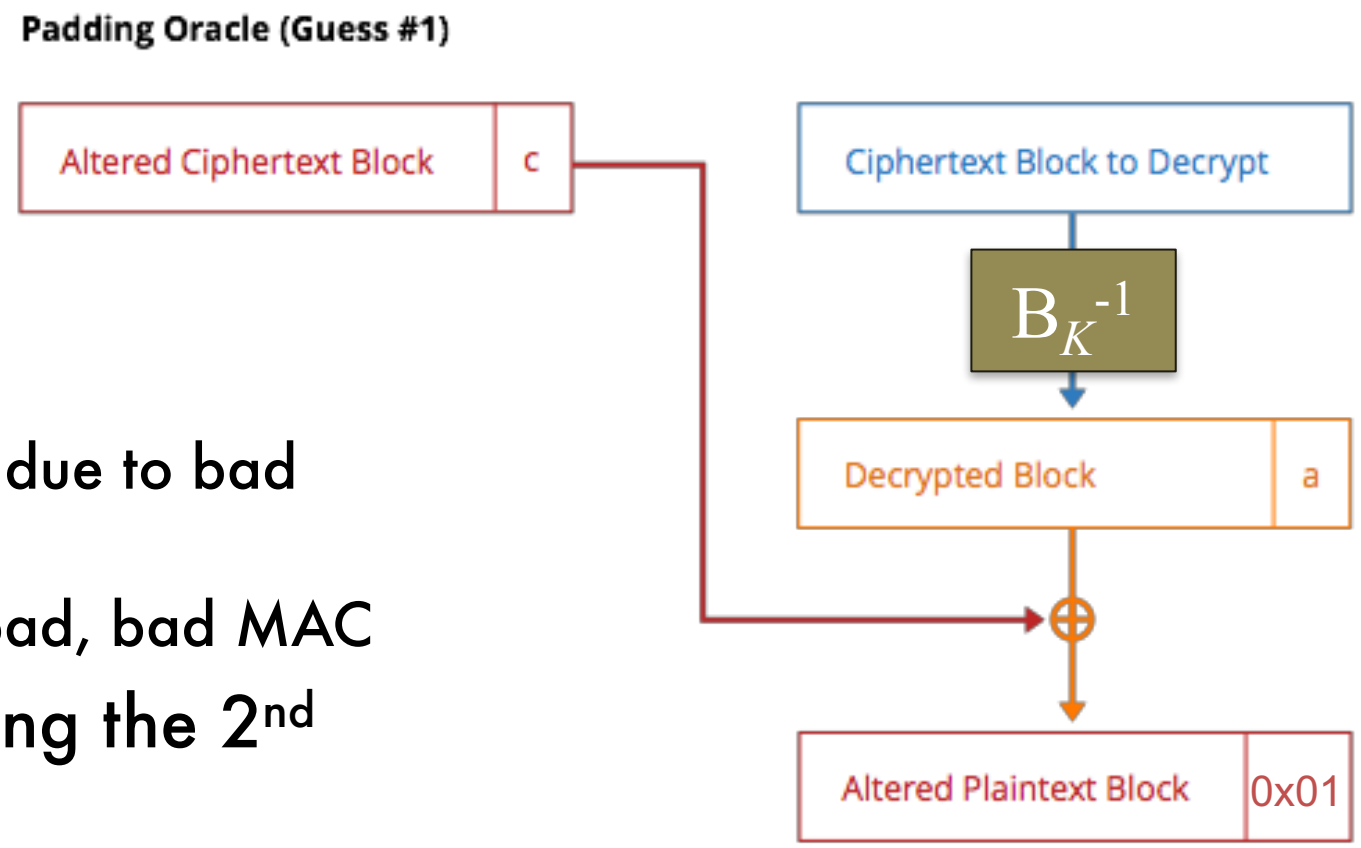
private P	tag (T' || 0303)	pad 01

- Different error messages → can distinguish between these two cases
- Use distinction to learn actual value of the final byte (here, 03)
- Rinse, lather, repeat!

Padding oracle attack: the execution

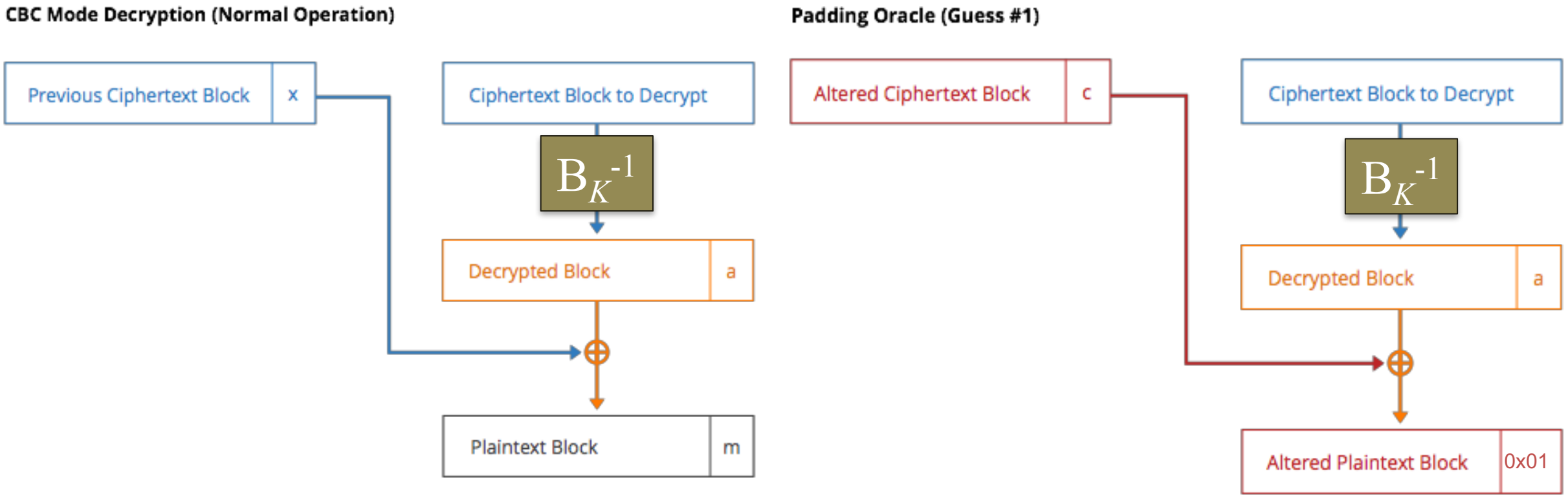
Attack procedure

- Send 256 CTs to Bob, one for each value of c
- Probably all will fail, return error messages
  - 255 of the failures will be due to bad padding
  - 1 failures will have valid pad, bad MAC
- Save the value of c causing the 2nd error!



CLOUDFLARE

Padding oracle attack: computing the message



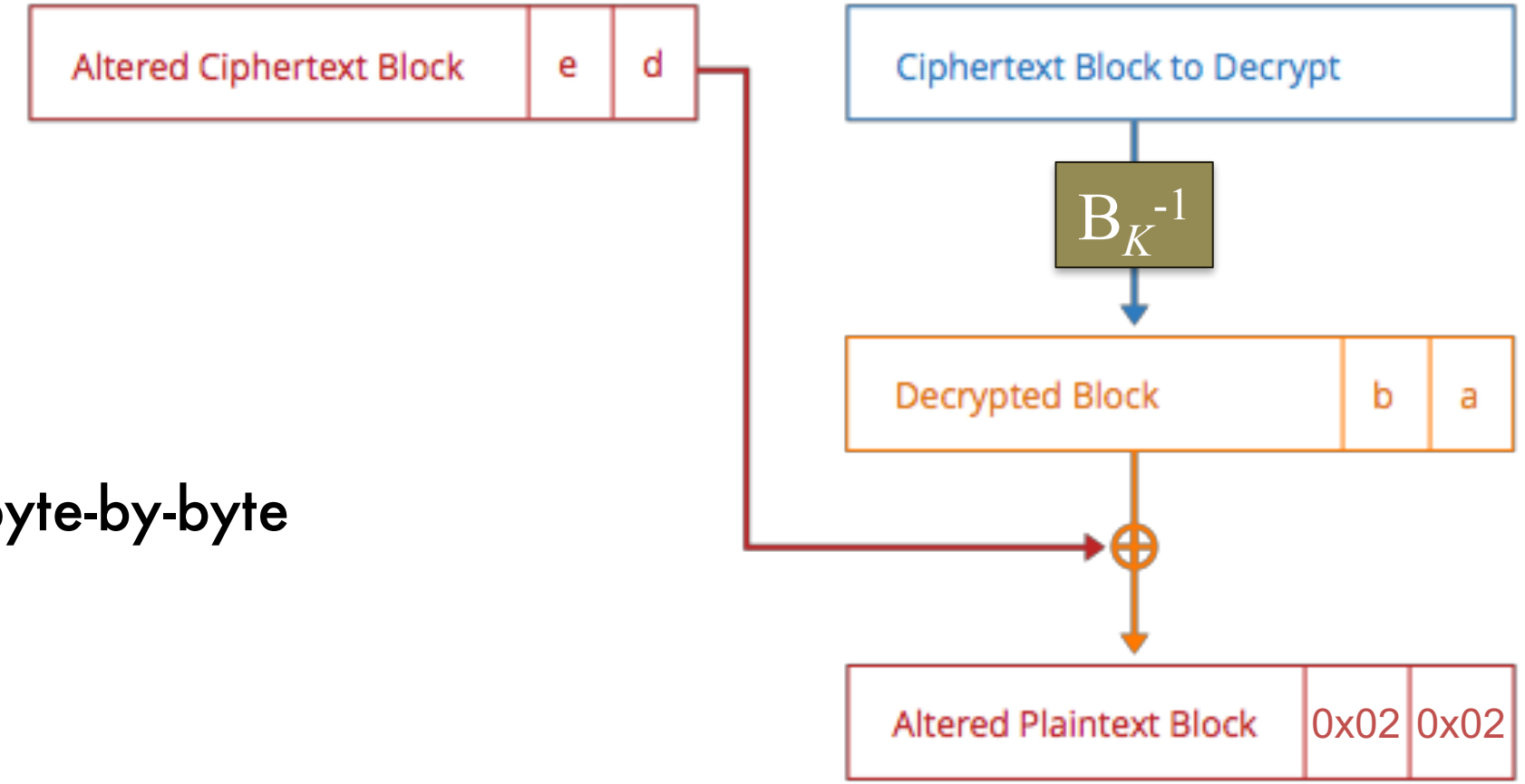
We can compute a in two ways: a = x ⊕ m = c ⊕ 0x01

So, the original message byte m = x ⊕ c ⊕ 0x01

CLOUDFLARE

Padding oracle attack: getting more bytes

Attack byte-by-byte



CLOUDFLARE

How can we fix this?

- Remember the three cases
  1. Invalid padding
  2. Valid padding, wrong HMAC
  3. Valid padding, right HMAC
- Required effort
  - ⇒ Read the padding bytes
  - ⇒ Read padding bytes, compute the HMAC
  - ⇒ Read padding bytes, compute the HMAC
- Bob's solution: return the *same* error message in cases #1 and #2
- Mallory's countermeasure: can still distinguish the two cases by observing the *time* that the MAC-then-Encrypt system takes to execute!
- Bob's new solution: ensure that crypto software's running time is *independent* of input; here, perform the HMAC test whether the padding is correct or not
- Mallory's new countermeasure: exploit timing variation within HMAC itself ☹

How to fix these vulnerabilities?

Can check to make sure you're operating on exactly what you expect

...but you had better make sure that this check is itself timing-independent

...and even then the fix might introduce a side-channel of its own

Basically, timing-independence is really hard!

OpenSSL Fact @OpenSSLFact Jul 24, 2013

/*The aim of right-shifting md_size is so that the compiler doesn't figure out that it can remove div_spoiler...which I hope is beyond it.*/

(So is software in general.)

OpenSSL Fact @OpenSSLFact Sep 3, 2012

/* [we should] obviate the ugly and illegal kludge in CRYPTO_mem_leaks_cb. Otherwise the code police will come and get us.*/

OpenSSL Fact @OpenSSLFact Jan 22, 2013

/* EEK! Experimental code starts */

OpenSSL Fact @OpenSSLFact Sep 5, 2012

/* BIG UGLY WARNING! This is so damn ugly I wanna puke ... ARGH! ARGH! ARGH! Let's get rid of this macro package. Please? */

How to fix these vulnerabilities?

(So is fighting against a compiler in general)

Mudge @dotMudge · Jan 25

Modern compilers make a lot of optimizations and perform advanced heuristics to determine what to emit. The resulting binaries have many (attack-able) components you cannot learn from the source alone.

Source is the intent, the binary is reality.

Steven Bellovin @SteveBellovin · Jan 25

My favorite is how hard it is to zero out a cryptographic key that you're done with--the optimizer says "this variable is never used again", so it deletes the zeroize operation.

Basically, timing-independence is really hard!

OpenSSL Fact @OpenSSLFact Jul 24, 2013

/*The aim of right-shifting md_size is so that the compiler doesn't figure out that it can remove div_spoiler...which I hope is beyond it.*/

(So is software in general.)

OpenSSL Fact @OpenSSLFact Sep 3, 2012

/* [we should] obviate the ugly and illegal kludge in CRYPTO_mem_leaks_cb. Otherwise the code police will come and get us.*/

OpenSSL Fact @OpenSSLFact Jan 22, 2013

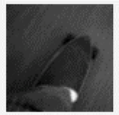
/* EEK! Experimental code starts */

OpenSSL Fact @OpenSSLFact Sep 5, 2012

/* BIG UGLY WARNING! This is so damn ugly I wanna puke ... ARGH! ARGH! ARGH! Let's get rid of this macro package. Please? */

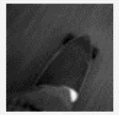
Padding Oracle Timeline

- 2002: Serge Vaudenay discovers CBC padding oracle attacks
- 2002-11: Extensions to specific systems like XML Encryption
- 2011: BEAST (Browser Exploit Against SSL/TLS) builds Java applet to perform the padding oracle in TLS 1.0
- 2013: Lucky 13 (TLS messages with 2 correct padding bytes processed faster than 1)
- 2014: POODLE (Padding Oracle On Downgraded Legacy Encryption) finds that the straightforward oracle works on SSL 3.0
- 2015: Extended Lucky 13 attack on Amazon's timing-independent TLS implementation
- 2017: TLS 1.3 breaks backward compatibility, permits Enc-then-MAC

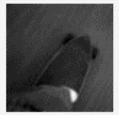


Jan Schaumann @jschauma
Attack timeline T given a theoretical vulnerability V:

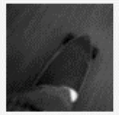
T0: academic research shows an attack is possible
Industry: Pfft, unrealistic.



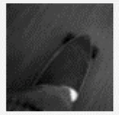
Jan Schaumann @jschauma
T1: nation-state attackers use the attack covertly
Industry: See, nobody's using this, nothing to worry about.



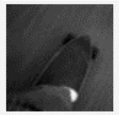
Jan Schaumann @jschauma
T2: academic research shows an attack is feasible with $$$$
Industry: We still have time.



Jan Schaumann @jschauma
T3: actually sophisticated attackers start using it
Industry: We should do something. Not today, but definitely Soon(tm).



Jan Schaumann @jschauma
T4: attacks become commodity, metasploit plugin appears
Industry: *gulp* *scrambles* *panic*



Jan Schaumann @jschauma
T5: with much pain, industry eliminates attack vector
Industry: yay, we're all good now

What can you do?

Use good *crypto* coding conventions

This page lists coding rules with for each a description of the problem addressed (with a concrete example of failure), and then one or more solutions (with example code snippets).

Contents [hide]

Validate code for timing independence

agl / ctgrind

Code | Issues 0 | Pull requests 1 | Projects 0 | Pulse | Graphs

Checking that functions are constant time with Valgrind

3 commits | 1 branch | 0 releases

Branch: master | New pull request

Adam Langley C++ support and constify pointers ...

Makefile	Initial import
README	A couple of typos
ctgrind.c	C++ support and constify pointers
ctgrind.h	C++ support and constify pointers
test.c	Initial import
valgrind.patch	Initial import

README

```
Checking that functions are constant time with Valgrind.
```

Related sources of error

Compression oracles

- Basic idea: if you apply compression before encryption, then post-compression message length reveals some information about message!
- 2012: CRIME (Compression Ratio Info-leak Made Easy) recovers secret web cookies over HTTPS connections, hijacks sessions
- 2013: BREACH (Browser Reconnaissance and Exfiltration via Adaptive Compression of Hypertext)

Format oracles

- Basic idea: if a higher-level protocol expects underlying message to obey some structural rules, can tinker with ciphertext until you find something that works
- 2011: "How to break XML encryption"
- 2015: "How to break XML encryption – automatically"

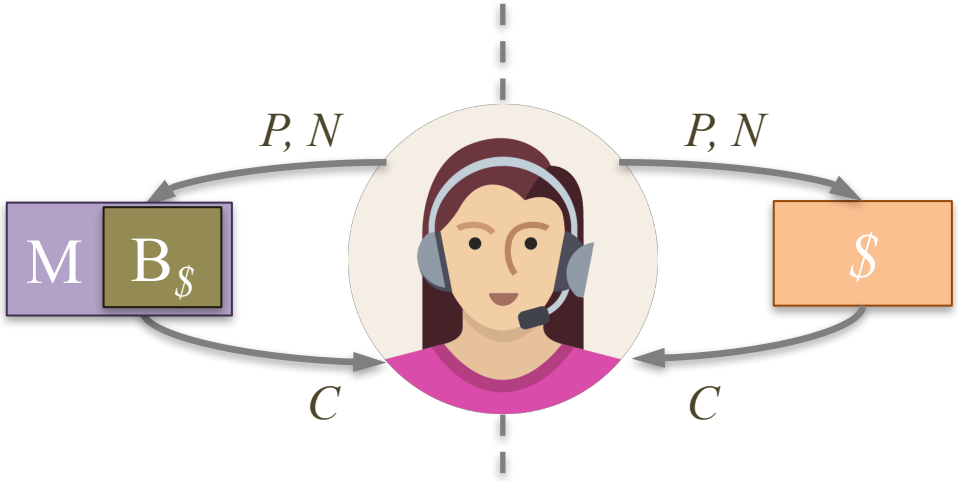
Summary: attacker oracles

- When we talk of 'oracles' in cryptanalysis, we mean that somehow the system is providing the attacker with the ability to compute f(P) for some function f
- There are many possible sources of these 'oracles'
  - Error messages
  - Message length, if a compression function is applied pre-encryption
  - Expected formatting of the underlying message (e.g., XML)
  - Time for a computation to finish
  - Performance speedup in running time due to the cache
  - Power consumed by the device

Part 1: privacy *XOR* authenticity

Privacy

IND$-CPA against nonce-respecting Eve

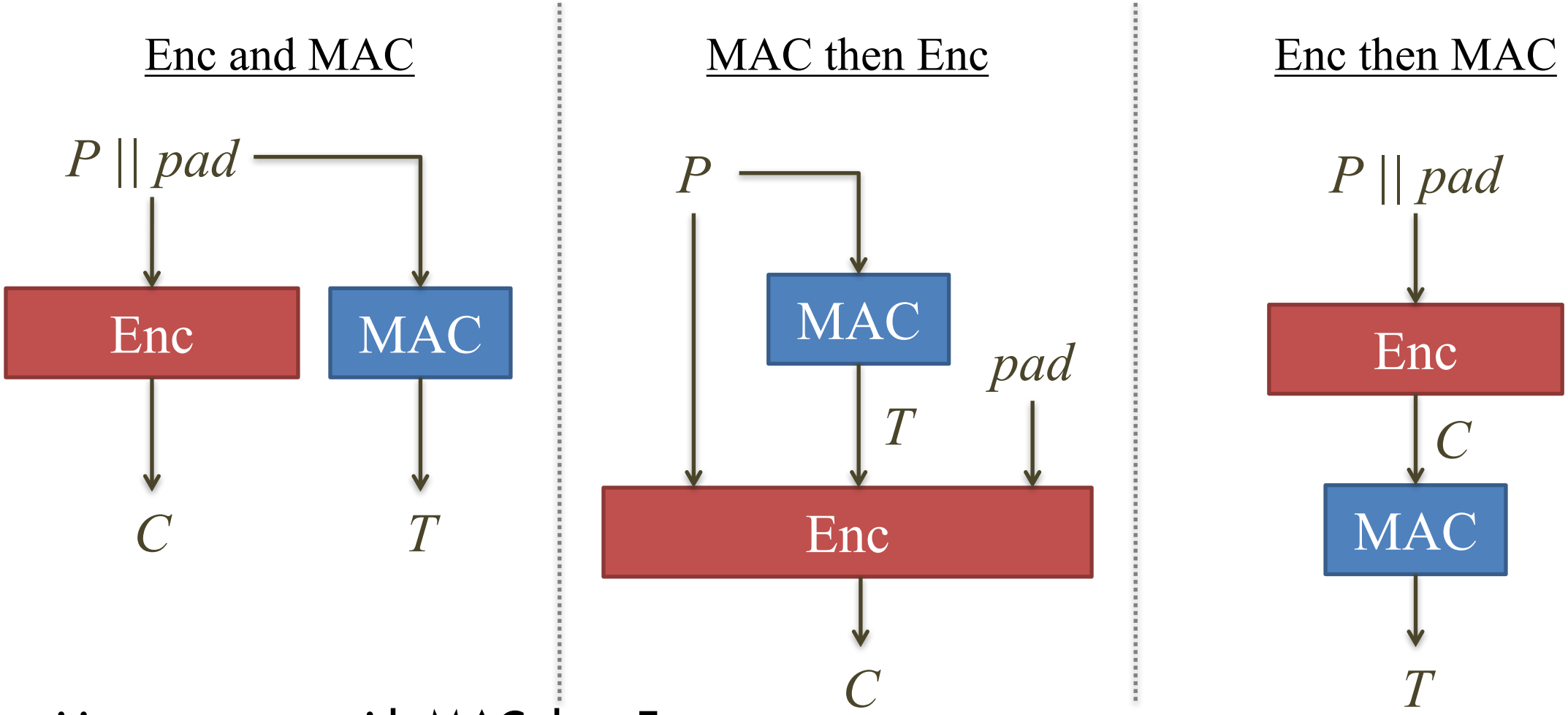


Authenticity

Even after viewing many (*A*, *T*) pairs, Mallory cannot forge a new one

Mallory wins if:
1. It's a valid forgery
2. It's new

How to combine Enc and MAC?



Intuitive concerns with MAC then Enc

- Recipient must perform decryption before knowing whether the message is authentic
- Leads to problems, as we just saw

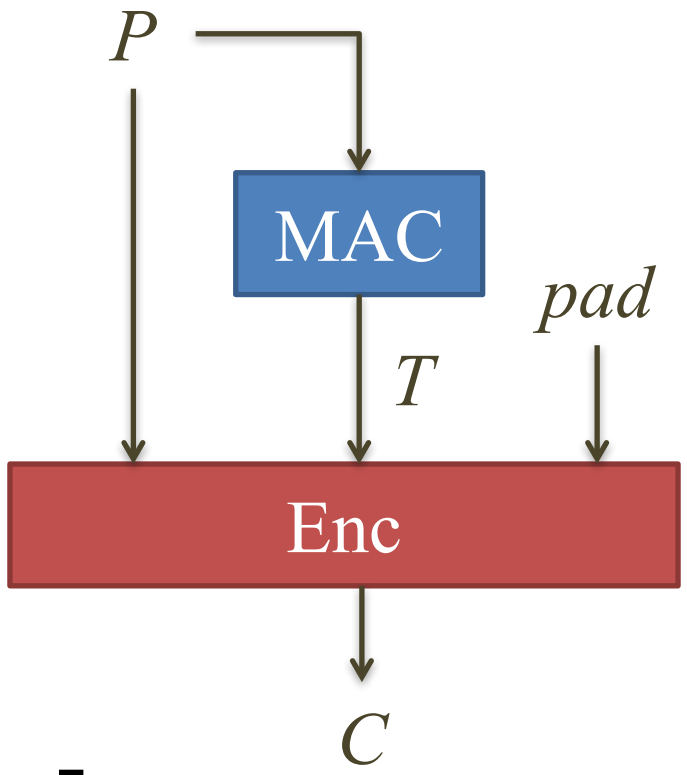
How to combine Enc and MAC?

Cryptographic doom principle

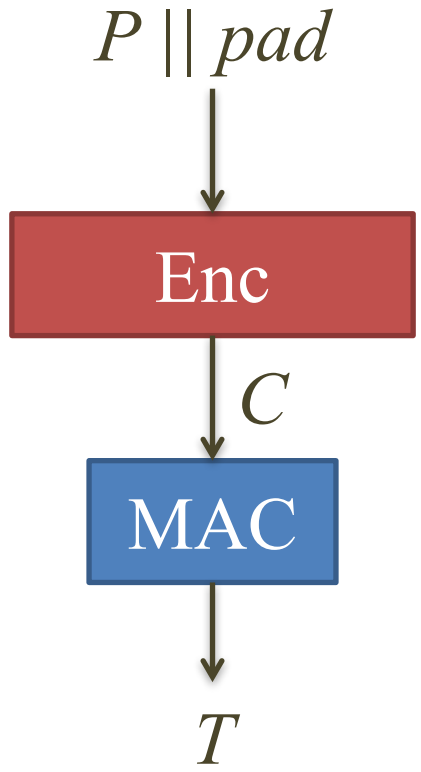
If you have to perform any crypto operation before verifying the MAC on a message you've received, it will somehow inevitably lead to doom!

– Moxie Marlinspike

MAC then Enc



Enc then MAC



Intuitive concerns with MAC then Enc

- Recipient must perform decryption before knowing whether the message is authentic
- Leads to problems, as we just saw

Next time: Authenticated Encryption

- Build toward *both* authenticity and privacy with the same construct!