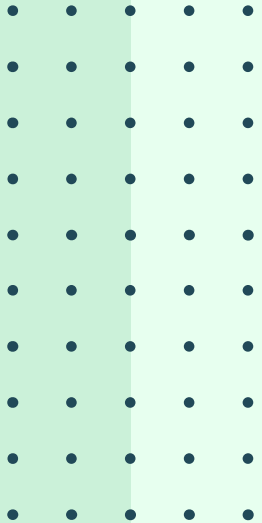
Gearup

Project 1: Cryptography



Overview

Attack some insecure “systems” of the fictional Blue University

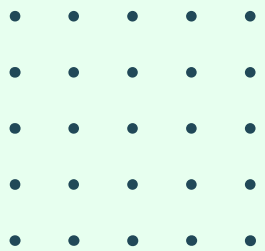
- Learn cryptographic principles, what attacks look like, & why you should never implement your own crypto (libraries are your friend!)

1660 students	1620/2660 students
1. Grades 2. Ivy 3. Passwords	Everything from Part 1 4. Padding

- Problems are self-contained & can be worked on in any order

Getting Started

```
- ...
|--DEV-ENVIRONMENT
| |--docker/
| |--home/
| | |--.etc/
| | |--p01-cryptography-yourname/  # <------------- Clone your stencil here!
| |--run-container
| |-- ...
...
```



Repository Layout

```
p01-cryptography-yourname
 |- grades/        # <--- Problem directory for ivy
 |  |- stencil/    # <--- Stencil code for grades
 |     |- go/
 |     |  |- STENCIL.md  # Guide for using this stencil
 |     |  |- sol.go
 |     |  |- ...
 |     |- python/
 |     |  |- STENCIL.md
 |     |  |- ...
 |     |- ...
 |- ivy/         # <--- Problem directory for ivy
 |  |- stencil/  # <--- Stencil code for ivy
 |     |- ...
 |- passwords/        # <--- Problem directory for passwords
 |  |- ...
 |- ...
```

Stencil Code

Stencils in Python & Go available for Grades, Ivy, & Padding

To begin, copy the stencil you want into the directory for that problem

```
cs1660-user@container:~/repo$ cp -Trv grades/stencil/python grades
```

STENCIL.md: read for helpful info about the stencil

(Go only) Makefile: run make to compile

Submitting

For every problem, your repo should have:

- Your solution program (usually sol)
- Any other required stencil files
- README
  - Describe the attack, how you did it, what you might change
  - See handout for per-problem details
  - Anything else we should know (what you tried, feedback, issues, etc.)

Your README is important—we're interested in your discussion/analysis!

Grades

You have:

- Database of grades, encrypted with ECB mode
  - Weak b/c the same plaintext block will always produce the same ciphertext block
- Some statistics
  - 100,000 students, 30 grades/student
  - Across all grades: 50% As, 30% Bs, …

You need to:

- Gather some information about the database, without decrypting anything

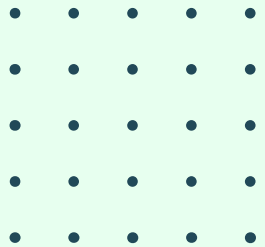
Types & Bytes

What type is a ciphertext? It's just bytes

```
# Get a string as bytes
str_as_bytes = "hello".encode("utf-8")  # b'hello'

# Construct arbitrary bytes
b = bytes([0xaa, 0xbb, 0xcc, 0xdd])  # b'\xaa\xbb\xcc\xdd

# Common to print in "hex-encoded" form
b.hex()  # 'aabbccdd'
str_as_bytes.hex()  # '68656c6f'
```



Ivy Wireless

You have:

- Encryption oracle: given plaintext *m*, returns *(iv, c)*
  - Can send as many plaintexts as want -> chosen plaintext attack
- Initial setup phase: client sends you encrypted key *k*

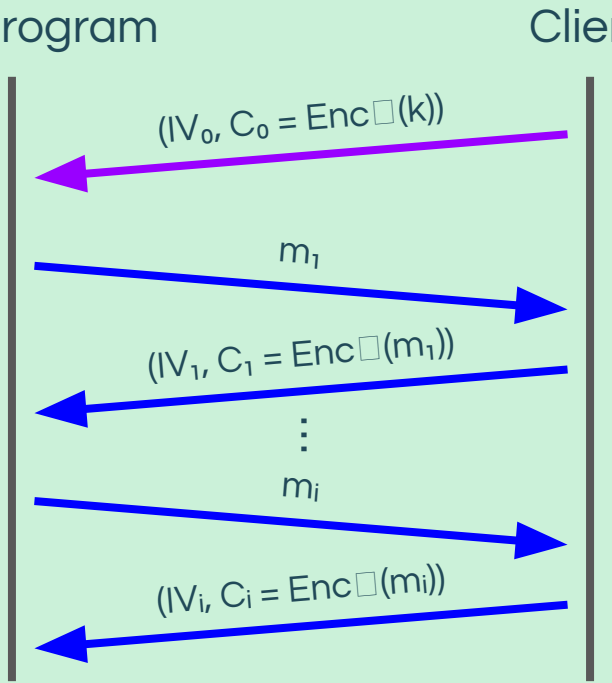
You need to:

- Recover key *k*

Interacting with Ivy Client

Setup phase: on startup, client sends encrypted key

If you can send as many plaintexts as you want, can you learn something about the key?



Normal operation: you send a plaintext message (hex-encoded string), client responds with a (IV, ciphertext) for that message

Passwords

You have:

- “Database” of passwords

You need to:

- Implement two methods of “secure” password storage
  - Single hash (sha1-nosalt)
  - Salted hash (sha1-salt4)
- Then attack!

Passwords are:

- 4 characters long
- Containing only lowercase ASCII letters (a-z) and digits (0-9)

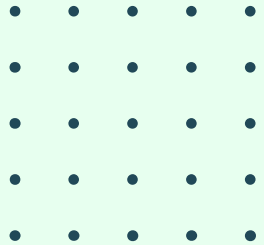
Hashing

More secure to store a hash of the user's password

```
{
  "method": "sha1-nosalt",
  "users": {
  "user3234": {
    "password": "1cc33637bdd3b586d89d259d719e8ad9a5e4f42e"
  }
}
```

But it can still be guessed, especially with a restrictive password policy

- What does that guessing look like?



Padding (CS1620/CS2660 only)

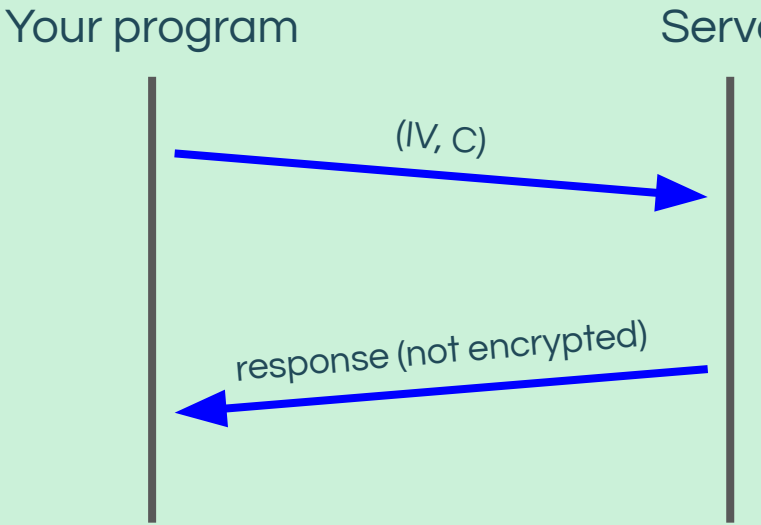
You have:

- "Grading server": given *(iv, c)* encrypted in CBC mode, returns plaintext *m* or error

You need to:

- Forge one *(iv, c)* pair that decrypts to a command that reveals student 12345's grades
  - There are many combinations of *(iv, c)* that do this—all you need to do is find one that works
- Not attempting to find key or break actual encryption process

Interacting with Grading Server



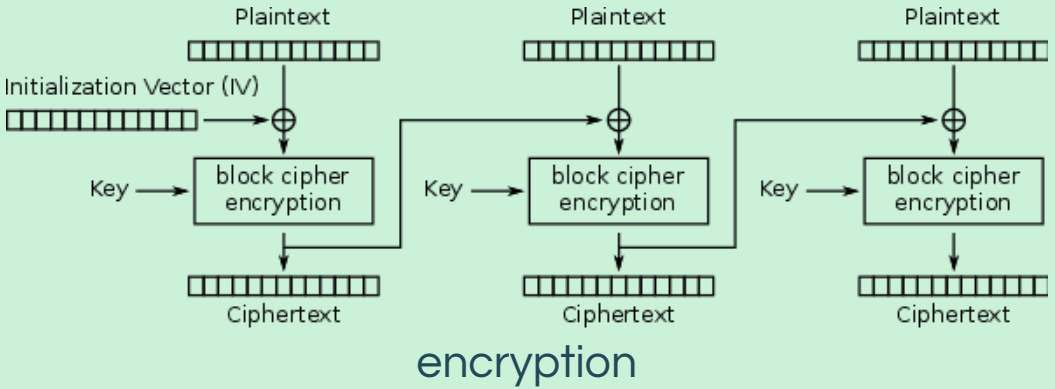
When server receives a message, it does something like this:

```
if len(c) not multiple of
block size (16 bytes):
  return error
m = decrypt(c)
if padding is invalid:
  return "incorrect padding"
result = run_command(m)
return result
```

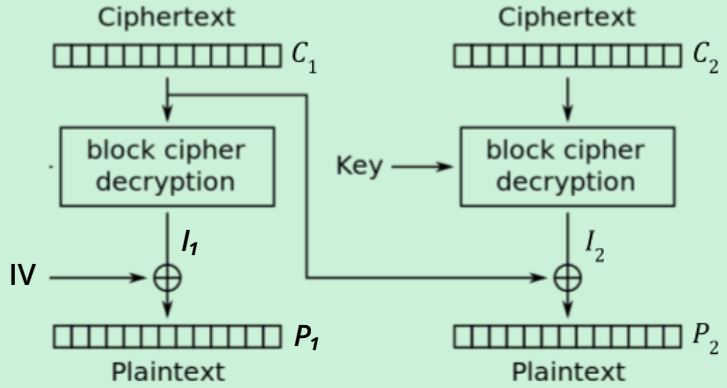
We can use the different kinds of errors to determine how far into breaking the system we got

result may be an error if invalid command

CBC Mode



encryption



decryption

Goal: choose (IV, C) to get desired command P = IV ⊕ I

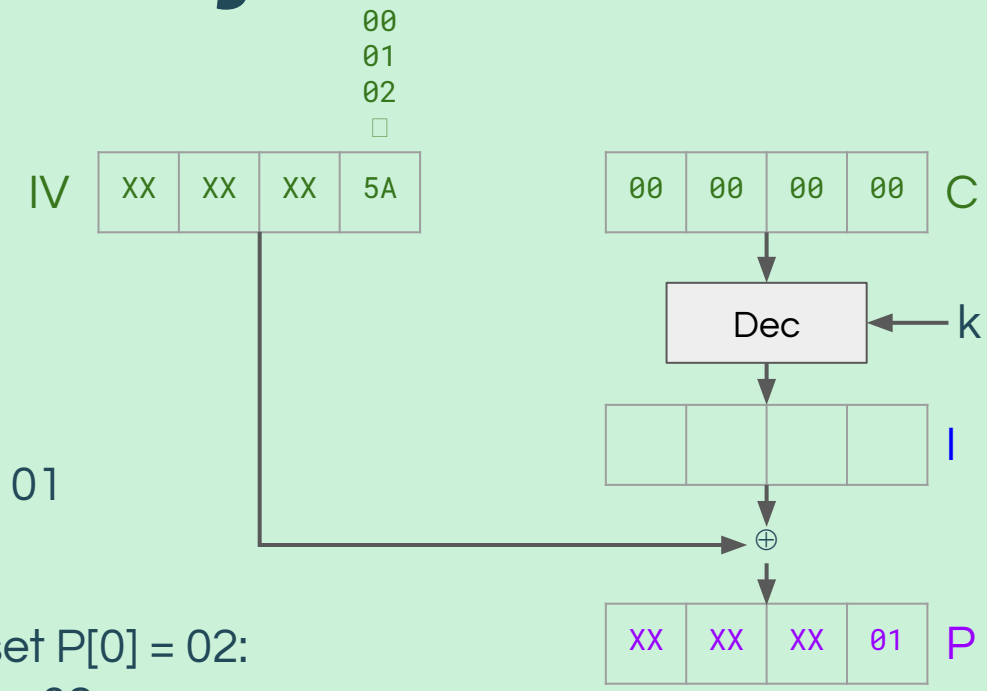
- Many combinations work, just need to find one
- IV is easier to control than C/I, so let's keep C constant
- Find I corresponding to constant C -> then can calculate IV for desired P

Work backwards across blocks with previous C = this IV

Finding I

Goal: find (IV, C) such that P ends in 0x01

- Different from IV to produce desired command P
- If we know IV (input) & P (padding leak), we can calculate I



$5A \oplus I[0] = 01$
$I[0] = 5B$

Then, to set P[0] = 02:
$IV[0] \oplus 5B = 02$
$IV[0] = 59$

Repeat to find all bytes of I