Cryptography

Due 02/14 at 11:59PM
### General Overview

<table>
<thead>
<tr>
<th>1660 (Part I)</th>
<th>1620 (Part II)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Grades</td>
<td>(Everything from part 1)</td>
</tr>
<tr>
<td>2. Ivy</td>
<td>1. Padding</td>
</tr>
<tr>
<td>3. Keys</td>
<td></td>
</tr>
</tbody>
</table>

All problems in each part are **separate/self-contained** → the assignment can be completed in any order.

*Move to the next problem if you get stuck!*
How the stencils work

- Each problem gets its own directory
- Stencil code in multiple language
- What lives in each stencil dir?
  - STENCIL.md: Super helpful stuff about this stencil
  - Makefile: If required => Run make to compile

To start: you should COPY the stencil files to the directory for that problem:

cs1660-user@container: ~/repo$ cp -Trv ivy/stencil/go ivy
This assignment is made up of a several small parts. This section describes how to navigate how we’ve organized the support and stencil code you’ll be using for the various problems, as well as some general mechanics about the project.

If you’re reading this document for the first time and want to see the problems, skim over this section and then return here when you’re ready to start writing code.

Repository setup

Each problem contains some support programs, as well as stencil code for developing your attack. You can download your starter code, and create a git repository you will use for development using this link: https://classroom.github.com/a/zhr68l_Z

After your clone your repository, the layout will look like this:

```
<repo root>
|- 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
  |  |- ...
- keys/      # <--- Problem directory for keys
  |- ...
- ...
```

Each problem has its own directory (the “problem directory”), which contains some support files and stencil code. Stencil code is provided in multiple languages (more on this in the next section).

Warning: Do not change this directory structure, as we expect your code to follow this layout when we clone your repository for grading. For more details on how to submit your code to work with our autograder, please read the “Assignment” and “Deliverables” sections for each problem.

Getting started

When you start on a problem, you should do the following:

1. Read over the problem’s section in this document, which provides background and describes how to run the support code
2. Look over the support code in your repository and decide which stencil you want to use. We have stencils for each problem in multiple languages (usually Python and Go). You can choose whichever stencil you feel most comfortable using for a particular problem.
3. Copy the files for your stencil of choice into the problem directory. For example, if you are working on the grades problem and want to use the Go stencil, you could run (from the repository root):

```
```
What you should submit

For each problem, your repo should have:

• Your script (called `sol`)
• (Anything else required by your stencil)
• README
  – Describe the attack, how you did it, what you might change
  – Anything else we should know (what you tried, feedback, issues, etc.)

At repo root: COLLABORATORS{,.txt,.md}
• List anyone you collaborated with, and on which part
1. Grades

→ GOAL: Uncover information from an encrypted database

→ YOU HAVE:
  a. Database encrypted with ECB mode
  b. Some statistics:
     a. 100,000 students total
     b. 30 grades/student
     c. Distribution of all grades: 50% As, 30% Bs, ...

● What do we want to know?

1. Since you know the plaintext format of the database, how many possible unique ciphertext blocks exist? (Hint: you can answer this question based on only the information here—then you can check your answer with a script!)


3. There's a student who's famous at the university for being the only student to ever get both As and Cs but no Bs. Exactly how many As, Cs, andNs has this student received?
Questions on grades

- If generate-database is slow, write file to /tmp (see Ed)
GOAL: Decrypt traffic from router

NEED: Shared (symmetric) key \( k \)

HAVE:

a. Ability to sniff network traffic

b. An “encryption oracle” via personal router \( \rightarrow \) chosen plaintext attack

c. The key \( (k) \) encrypted \( \rightarrow E_k(k) \)
Ivy demo
Ivy (cont.)

**ENCRYPT**($k$, $m$)

1. $iv = R()$  
   // Generate initialization vector
2. $s = iv + k$  
   // Concatenate $iv$ and $k$
3. $r = G(s, |m|)$  
   // Generate $|m|$ random bytes
4. $c = m \oplus r$  
   // XOR $m$ and $r$ to get ciphertext $c$
5. **return** $(iv, c)$

**DECRIPT**($k$, $iv$, $c$)

1. $s = iv + k$  
   // Concatenate $iv$ and $k$
2. $r = G(s, |c|)$  
   // Generate $|c|$ random bytes
3. $m = c \oplus r$  
   // XOR $c$ and $r$ to get plaintext $m$
4. **return** $m$

- $R$ is a source of randomness. Each time $R$ is queried, it generates a uniformly-distributed random number which is 16 bits long (that is, its outputs are uniformly distributed in $\{0,...,2^{16} - 1\}$).
Keys

→ **GOAL:** Break Academy’s block cipher encryption scheme
→ **NEED:** Both encryption keys
→ **HAVE:**
  a. Set of (plaintext, ciphertext) pairs from encryption scheme
  b. Cipher + stencil code

\[
E_k(m) = c \quad \text{[encryption]} \\
D_k(c) = m \quad \text{[decryption]}
\]

\[
E_{k_1}(m) = c' \quad \rightarrow \quad E_{k_2}(c') = c \\
D_{k_2}(c) = c' \quad \rightarrow \quad D_{k_1}(c') = m
\]
Questions on Keys

- **Java stencil: you must install Java first**
  - See STENCIL.md for instructions

- **Performance?**
  - No specific requirements, so long as you carry out the attack we’re asking you to perform
  - Autograder will time out at 40min—which should be much more than enough
  - You can discuss performance, time/space complexity in your README!

- **What’s the cipher?** Irrelevant to problem (but look up TEA cipher)
GOAL: Reveal the grades of student with ID: 12345

HAVE:
   i. Server as a binary
   ii. Encryption scheme → CBC mode
   iii. Whether the padding is correct or not

NEED: Ability to find correct ciphertext given any plaintext:
   i. Determine current intermediate state $I_n$
   ii. Find correct previous ciphertext $C_{n-1}$ to generate $P_n$
GOAL: Forge your transcript

NEED: (?)

HAVE:

a. Challenge/response protocol binary ("challenge")
b. JSON-encoded copy of the website’s public key ("server.pub")
c. Signature method ("encrypt") via binary
d. Knowledge that the same RSA key pair is used for signatures & verification of the website’s integrity