CIS 4930 – Intro to Cryptography  Spring 2021

Homework 4: Deadline Friday 3/26
Instructor: Viet Tung Hoang

Acknowledgment: I am indebted to the help of Nickolas Donnell in setting up the testing server. Nick also generously provided some example code for how to interact with the server.

Goal. In this homework (which is worth 150 points), you are asked to implement the padding-oracle attack. The target server is linprog7.cs.fsu.edu (IP address is 128.186.120.191), and the port number is 31337. Due to security restriction by the CS department, to have a socket connection to this server, you need to run your code at the same machine. In order to log in to this machine, connect to linprog as usual, and then run the command “ssh user@linprog7.cs.fsu.edu” (replace “user” with your CS username). You only need to upload your files to linprog; those will be accessible from linprog7.

The server maintains a secret string \( M \) which your program needs to recover. Currently, this string is (without the quote and case sensitive) “Congratulation! You know ur crypt0!!! : )”, and thus the byte length is 48. However, when I test your programs, I may change it to a different string, of a different length. You should not assume that the byte length of the secret is a multiple of 16.

Interacting with the server. The server provides two types of requests. If you want an encryption of \( P \| M \), for some chosen prefix \( P \), then you need to send a request of the form “-e \( P \)”. The server will send you a ciphertext \( C \) of \( P \| M \) under the TLS 1.0 encryption on a random IV \( L \). If you want to send a ciphertext core \( C \) with IV \( L \) for validation, you need to send a request of the form “-v \( C \) \( L \)”, and the answer is either “Valid” or “Invalid”. Your goal is to recover the message \( M \), and output it to stdout.

For an example of how to interact with the server, see the Python script client.py in the course website. When I run python3 client.py in my terminal, the script waits for me to enter requests to the server. If I type “-e abcdef0123456789” (meaning my prefix is the hexadecimal string abcdef0123456789), I get back b'Encryption: 80\n49 8d 8e 57 ...6a 3d de \nIV: b'a524e79b9f9b4f5634a8b00e06e46c' 
-E abcdef0123456789

Here the ciphertext core \( C \) (in hex encoding) is 49 8d 8e 57 ...6a 3d de that is 80-byte long, and the IV \( L \) (in hex encoding) is a524e79b9f9b4f5634a8b00e06e46c If you want to encrypt with the prefix as the empty string, just use “-E”. Likewise, if I type “-v aaaaaa 40beed9c1b5fbc997bc025a42b4c0a”, I get back ‘Invalid’. This means that the ciphertext core aaaaaa with IV 40beed9c1b5fbc997bc025a42b4c0a is not a valid ciphertext.

For another example of how to write code to connect with the server, see the file connect.py in the course website. (The latter is written in Python2, so you’ll need to run python connect.py.)

Another testing server. To help you to test your program on fragmentary secrets, I also set up another server at the same IP address, but the port is now 31336, and the secret is “Hello World”.

Deliverables. Upload to Canvas a zip file containing your source code, which includes a README.txt that informs me how I should run the program.