CS 409 : Introduction to Cryptography Fall 2025

Lab 2 — Block-Cipher Modes

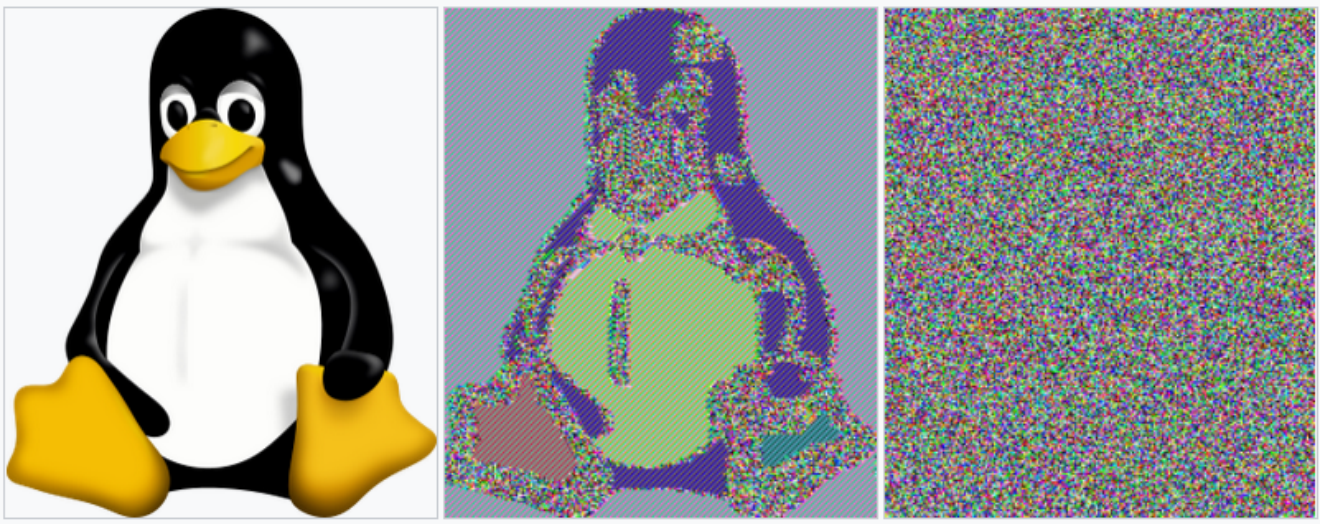
Instructor: Sruthi Sekar *TAs:* Ravi Prakash, Lakshya Gadhwal

Challenge 1: `openssl` Decryption

You've been provided a file `ciphertext.bin` which contains an encryption of the flag encrypted using `aes-128-cbc` with the key given in `key.hex` and the IV given in `iv.hex`. Use your knowledge of `openssl` to decrypt the ciphertext and retrieve the flag!

Challenge 2: The Electron Code

The ECB (Electronic Code Book) mode of encryption is a "bad" mode of encryption because it always maps the same block of plaintext to the same block of ciphertext. Thus, if you have a long message such that the same block repeats twice in the plaintext, the same block would repeat in the exact same position in the ciphertext too. This leaks information about the plaintext.
As an example of the amount of information this leak can reveal, here's a picture of what happens when you encrypt the famous Tux penguin using ECB mode:



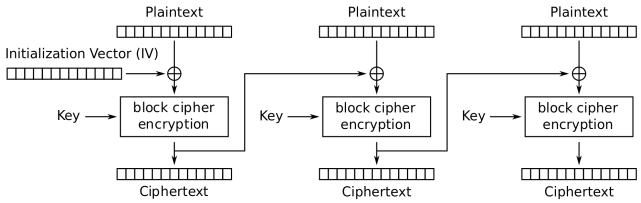
Original image | Using ECB allows patterns to be easily discerned | Modes other than ECB result in pseudo-randomness

In this challenge, you will exploit this redundancy of ECB to obtain the flag. The encryption code is present in `encryptor.py` and the resultant ciphertext is in `ciphertext.bin`.

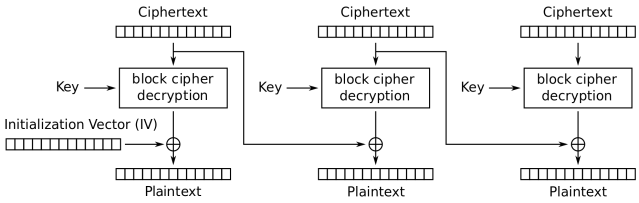
Challenge 3: The Catastrophic Equality

Some people believe that since the key is already being randomly generated, the key can also be repurposed as the IV as long as the IV is not being revealed. This is a rookie mistake and in general, a very bad idea.
In this challenge, you are going to submit "parameters" (each message is a string of the form *key1*=*value1*&*key2*=*value2*&$\cdots$&*key-n*=*value-n*) to a server. These messages are supposed to comprise of normal low-ASCII characters and should not contain the string `admin=true` as a substring.

You will receive the encryption of your messages. The server also lets you send encrypted parameters to it, which it will decrypt internally; while the server does not send you the decryption, it will throw an error if it finds offending invalid characters in the decrypted text. If you can somehow send the server an encrypted parameter string which decrypts to a normal string which has `admin=true` as a substring, you will receive the encryption of the flag in return.
Abuse the fact that the server repurposes the key as the IV to craft an intelligent payload which allows you to recover the flag. Good luck on your mission!
The following diagrams could be of help to you (here "encryption" and "decryption" refer to the forward and reverse applications of the block cipher):



Cipher Block Chaining (CBC) mode encryption



Cipher Block Chaining (CBC) mode decryption

Challenge 4: Never Painted by the Numbers

The CTR (CounTeR) mode of encryption is commonly used to convert a block cipher into an equivalent stream cipher. Do take a look at https://pycryptodome.readthedocs.io/en/latest/src/cipher/classic.html#ctr-mode to understand how to use the PyCryptodome library for CTR encryption mode (along with an explanation of how CTR mode works if you need a refresher).
In this challenge, you will interact with a server which essentially acts as an echo server - it outputs back to you exactly what you input to it – except that the response is encrypted. However, if you input `!flag` to it, the server instead outputs the flag to you – again, encrypted. The encryption is done using CTR mode, using a key you don't know. But you suspect that the implementation of the CTR mode encryption in the server has some vulnerability. Refer to the `server.py` template file for details on the workings of the server, identify the vulnerability and exploit it to retrieve the flag.

Bonus Challenge: Canis Lupus Familiaris

A padding oracle attack is a cryptographic attack that targets the padding scheme used in block cipher modes, particularly in Cipher Block Chaining (CBC) mode.
In practice, padding is often added to plaintext messages to ensure that the length of the message is a multiple of the block size of the cipher being used. A padding oracle is a term used to describe a vulnerability that arises when an attacker can determine whether a given ciphertext has a valid padding or not. The padding is typically added to ensure that the plaintext can be properly aligned into blocks before encryption. The padding scheme we often use is PKCS#7.
The padding oracle attack can be extremely dangerous in the sense that one could completely recover the plaintext given access to a padding oracle. In fact, that is what you will do in this challenge. You can refer to this webpage to learn about the technical workings of the padding oracle attack. You would be familiar with a lot of the material in the initial part of the webpage, so you can directly start reading from the section titled "Cipher-block chaining (CBC)".
In this challenge, you will interact with a server which will act as a padding oracle. In `solution_template.py`, you can make use of the function `validate_padding` as a blackbox for server communication (refer to the script comments for further details).

Historical Note: *The padding oracle attack forms the basis for a nasty attack, famously referred to in the field of network security as the POODLE attack. POODLE, which stands for "Padding Oracle On Downgraded Legacy Encryption", is a security vulnerability that affects the widely used SSL (Secure Sockets Layer) and early TLS (Transport Layer Security) protocols. This attack was discovered in 2014 and is a serious threat to the security of encrypted communications.*
The primary consequence of POODLE is the potential disclosure of sensitive information. By successfully executing the attack, an attacker can decrypt and access the contents of encrypted communication, including login credentials, personal information, and other sensitive data.
Web browsers often support multiple SSL/TLS versions to maintain compatibility with various websites. POODLE can be particularly problematic for browsers, as attackers can exploit the vulnerability to compromise the security of connections made by the browser.
The discovery of POODLE accelerated the process of deprecating and phasing out support for SSL 3.0 across the industry. Subsequent versions of SSL and TLS, such as TLS 1.0, TLS 1.1 and TLS 1.2, have also been deprecated due to security vulnerabilities.