AES is a **block cipher encryption algorithm** that encrypts **a fixed size of data**. Let's say we have the following plaintext that is 30 bytes long we will encrypt using AES-128.

```
AES encryption is pretty cool.
```

The **128** in AES-128 signals that the block size is 128 bits (16 bytes) long, so the above plaintext will be broken into following blocks before being encrypted.

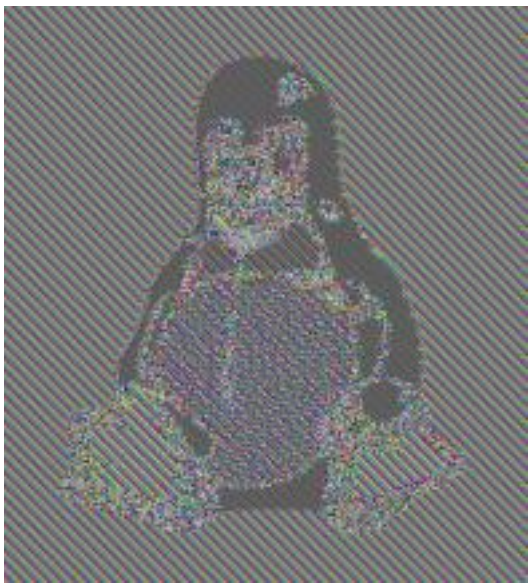
Block 1

```
AES encryption i
```

Block 2

```
s pretty cool.
```

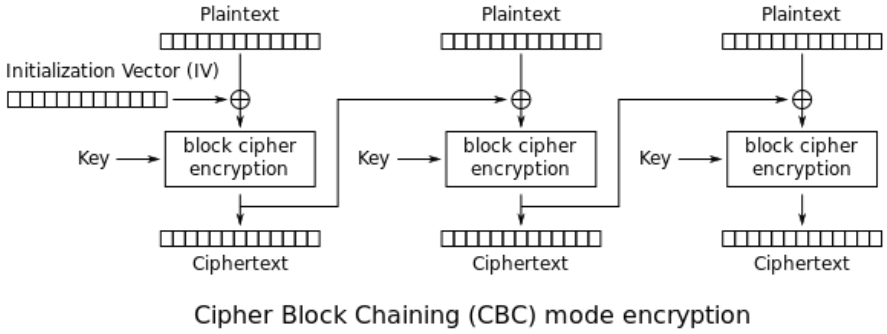
However, just using AES encryption for data that has repeated values is **insecure** because two blocks with the same plaintext will have the same ciphertext. The following *encrypted* image of the Linux penguin is an example of where it is insecure by showing that the *encrypted image* still reveals the outline of the original image.



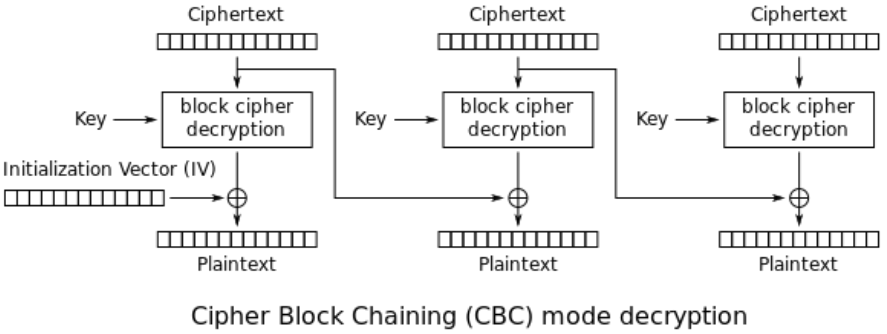
This is why **cipher block modes of operation** are used around AES to prevent repeated blocks having the same ciphertext.

This challenge uses the **Cipher Block Chaining (CBC) mode of operation** for encrypting the `auth_cookie`. CBC mode works by using the encryption output of the previous block and **XORing** with the plaintext for next block before that block is encrypted using AES. An **initial vector (IV)** needs to be set when using CBC mode to encrypt the first block, since there are no previous

blocks when encrypting the first block. The following diagrams show how CBC mode works for encrypting and decrypting data with multiple blocks.



CBC mode encryption



CBC mode decryption

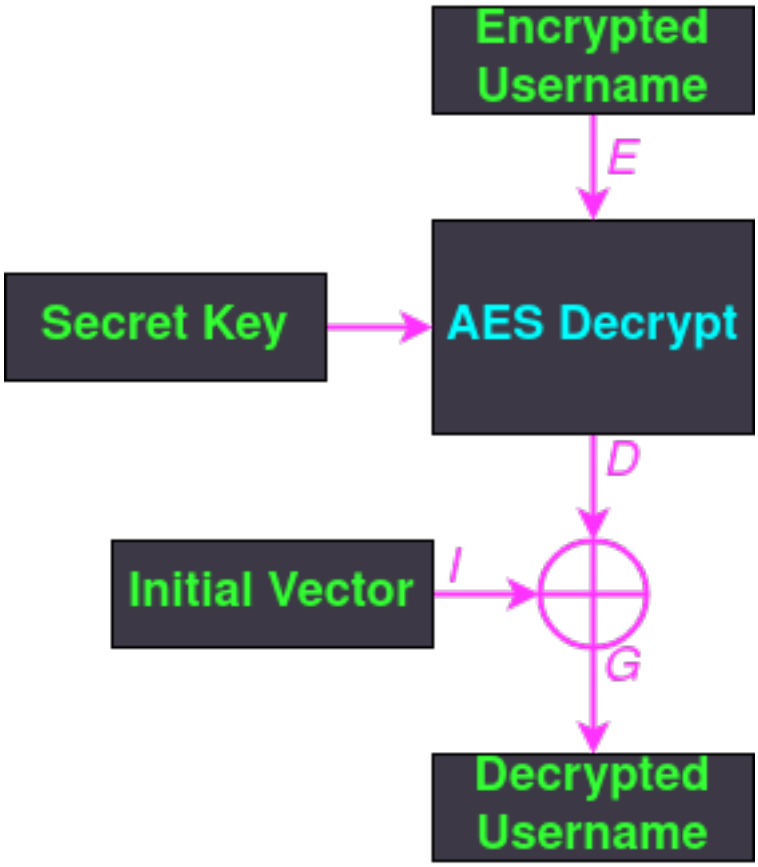
If you are still confused about cipher block modes and CBC mode, I highly recommend watching this Computerphile video.

For this challenge, you need to focus on **CBC decryption** since that is what we want to attack. Below are the following things that we know and have access to:

- The format of the `auth_cookie` is `{initial_vector}:{encrypted_username}` where `{initial_vector}` (I) is the initial vector and `{encrypted_username}` is the encrypted username (G).
- The web server authenticates users by grabbing the initial vector and encrypted username from the `auth_cookie`, decrypts the username and checks the username.
- You have access to the `guest` account, so when you first login as the `guest` account the `auth_cookie` that is set for your session will decrypt to the value of `guest` (G) on the web server.
- You want to **trick the website to decrypt the username to `admin` instead of `guest`**.

The following diagram visualises how the web server authenticates users using CBC mode. You only need to worry about a single block because the usernames

for this challenge are less than 16 bytes long.



Of particular interest for this challenge is the **XOR operation that is used after AES decryption**. Let's say that the AES decrypted value is D, the initial vector is I and we know that when you login with the `guest` account the decrypted username (G) will be `guest`. The following mathematical equation shows this XOR operation where XOR is denoted as $\oplus$.

$D \oplus I = G$

The above equation is all you need to solve this challenge and a basic understanding about the **mathematical properties of XOR**.

You need to figure out a way to a way to manipulate the above XOR equation so the result is the username `admin` instead of `guest`.

Focus on things in the above equation that you can manipulate.