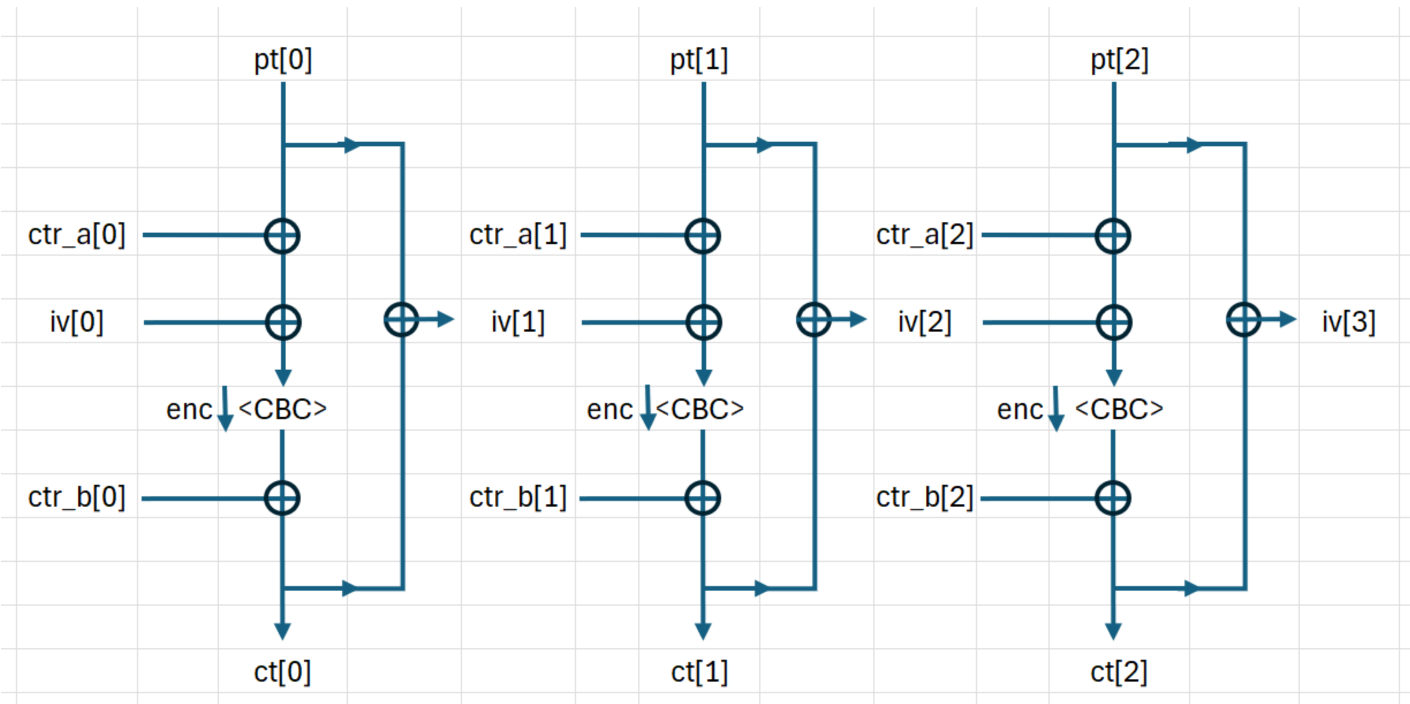
pcbc revenge - crypto

tl;dr hhhhh (https://github.com/DownUnderCTF/Challenges_2023_Public/tree/main/crypto/hhhhh)-like block swapping and linear system solving

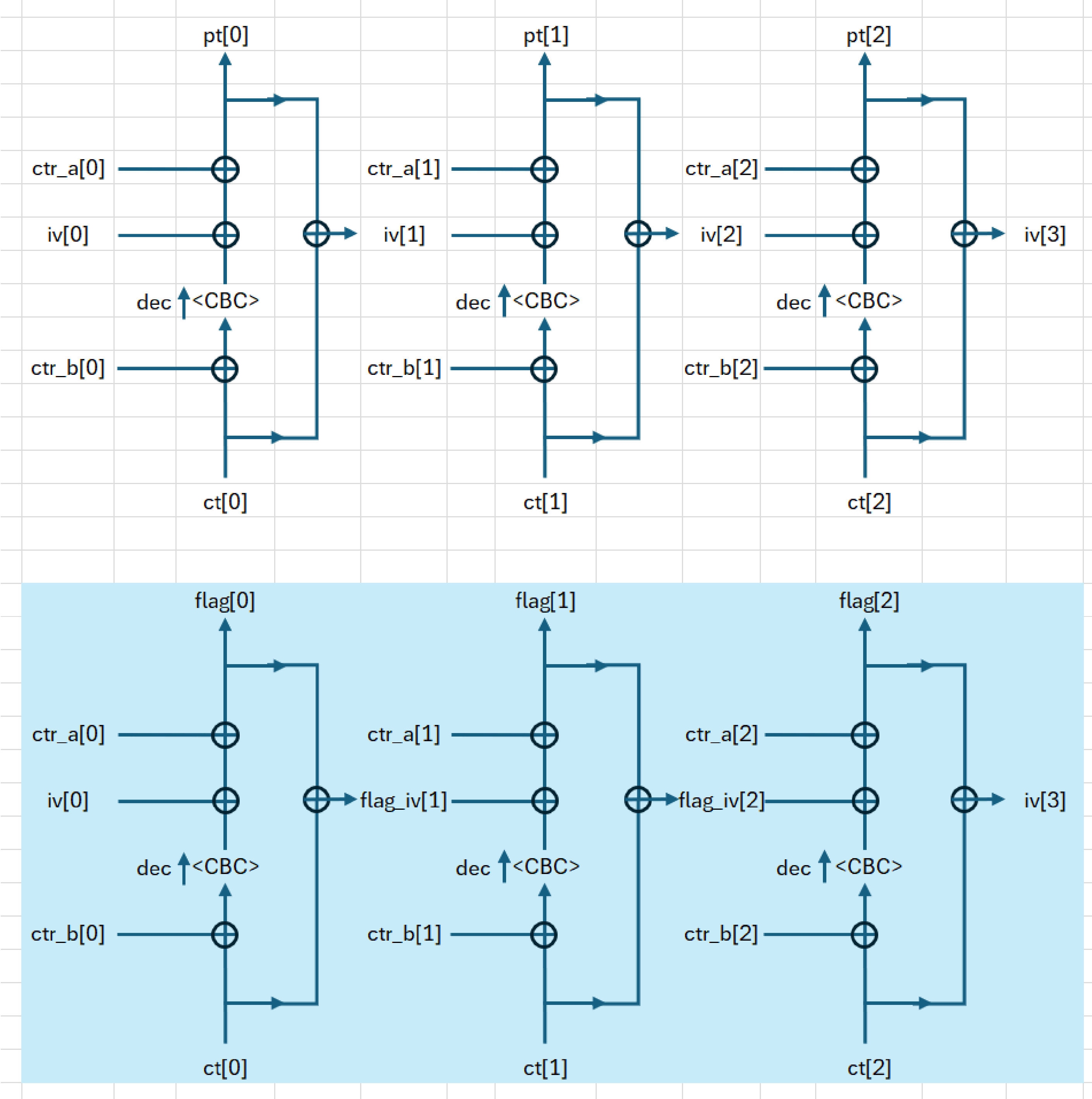
There is an implementaion of a cursed block cipher with some CTR and some CBC behaviour. We can obtain `(pt, ct)` pairs by querying with length of desired pt but we have no further control over the contents of `pt`. There is also a decryption padding oracle.

Let's model the data flows between blocks, treating the two CTRs as one-time-pads:

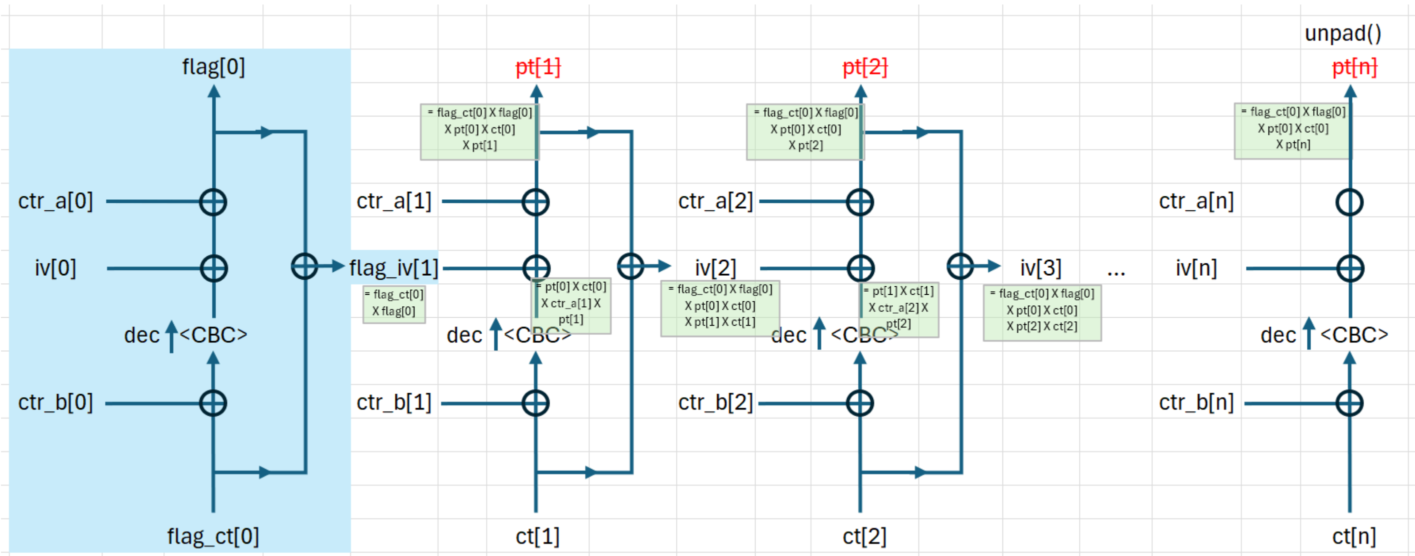
Encryption operation



Decryption operation



"Block swapping"



We can request a large `(pt, ct)` pair and then replace it's first (non-iv) block with the flag encryption. The result is that a `flag[0] ^ flag_ct[0]` term will be propagated throughout the cipher up until affecting the decryption output of the last block.

In order to gain more control over `iv[n]`, we can make use of a second large `(pt', ct')` pair. Each time we exchange one block of `ct[i]` with `ct'[i]`, the effect is `iv[n]` is XOR-ed with

```
pt[i]   ^ ct[i]   ^ pt'[i]   ^ ct'[i]   ^
pt[i-1] ^ ct[i-1] ^ pt'[i-1] ^ ct'[i-1]
```

Given long enough `(pt, ct)` pairs with at least 128 blocks each, we use a system of linear equations over $\mathbb{F}_2$ to specify the desired `iv[n]`; the solution to the system is then used to decide when to use chunks from `ct` or `ct'`. This reduces to a typical padding oracle attack which allows us to solve for `flag[0] ^ flag_ct[0]`, and since we know `flag_ct` we can recover `flag`.

Later chunks in `flag_ct` can be recovered by using the first 2 blocks, then 3 blocks and so on until the flag encryption is exhausted.