- Side-channel attack
- Chosen ciphertext attack
- Can be used to decrypt any ciphertext without knowing the key
- Can also be used to encrypt any plaintext without knowing the key

You need a system where:

- The ciphertext is modifiable by you
- Uses CBC mode for multiple blocks
- Uses PKCS#7 as the padding mechanism
- For encryption: the IV is modifiable by you
- The system returns for a given ciphertext whether the padding is okay or not before parsing the message

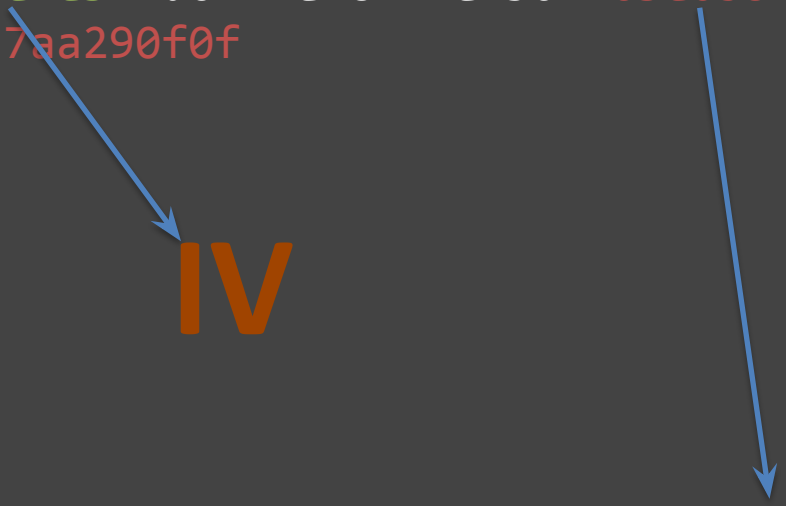
```
curl  "http://<url>/token?token=794ebcc5bd4e4e314d6447e4cf1fef50
       4163e0b8f013990f81460d387d36ed9d781d3bd7aa290f0f"
Ooops! Padding problem
```

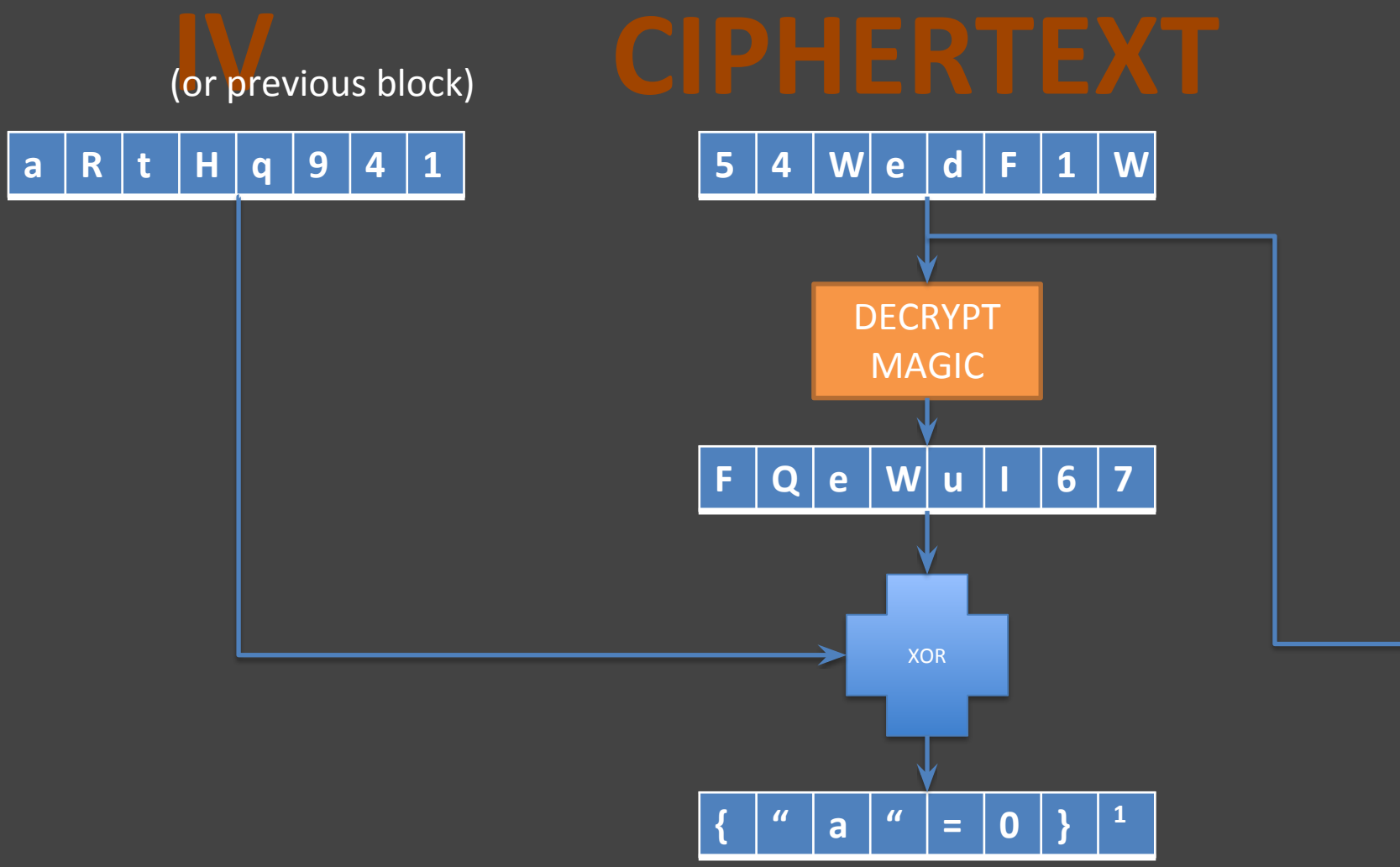
Other common ways:

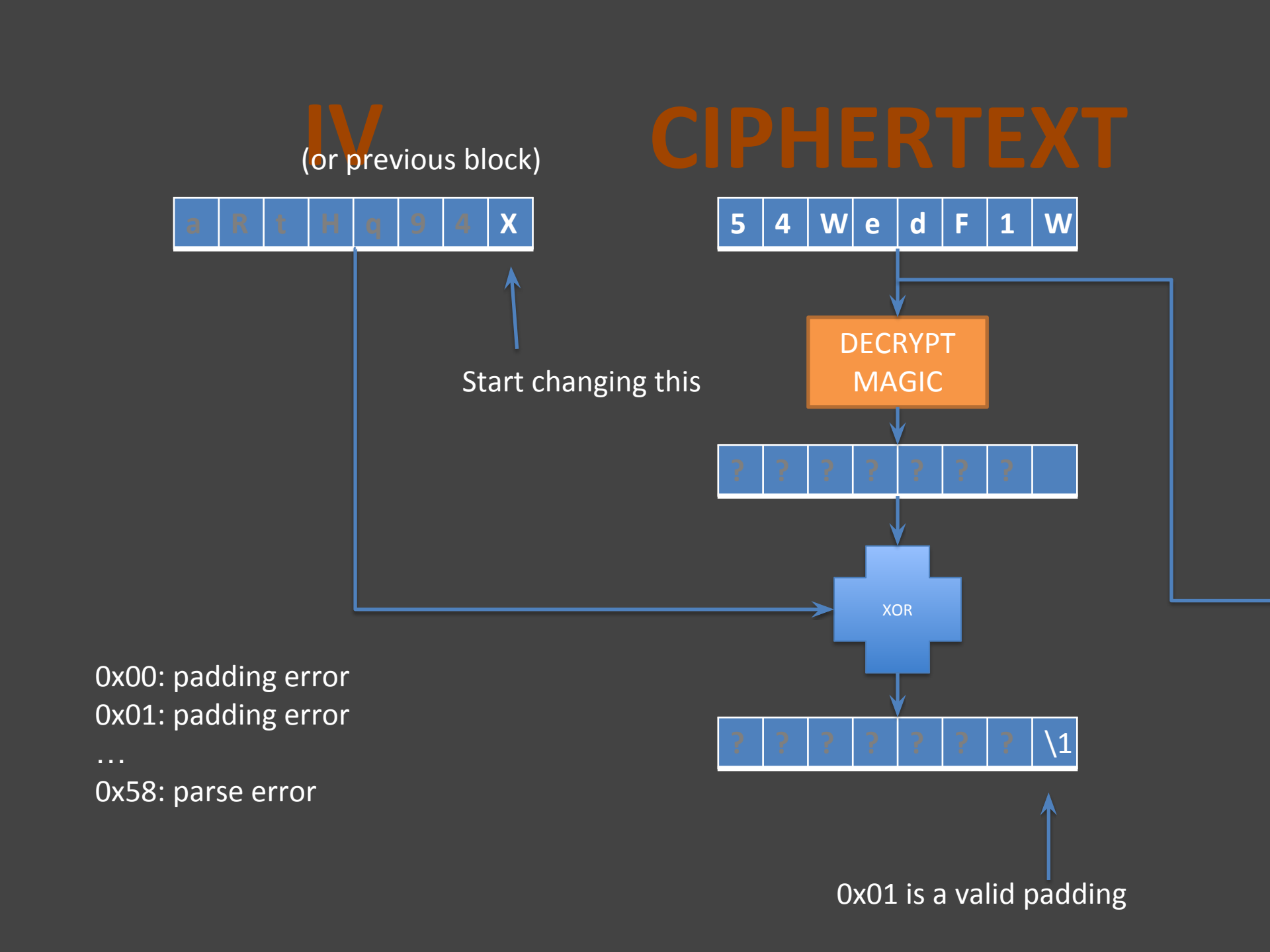
- Different status codes
  - 200: OK
  - 422: Parse error
  - 500: Wrong padding
- Using a different side channel

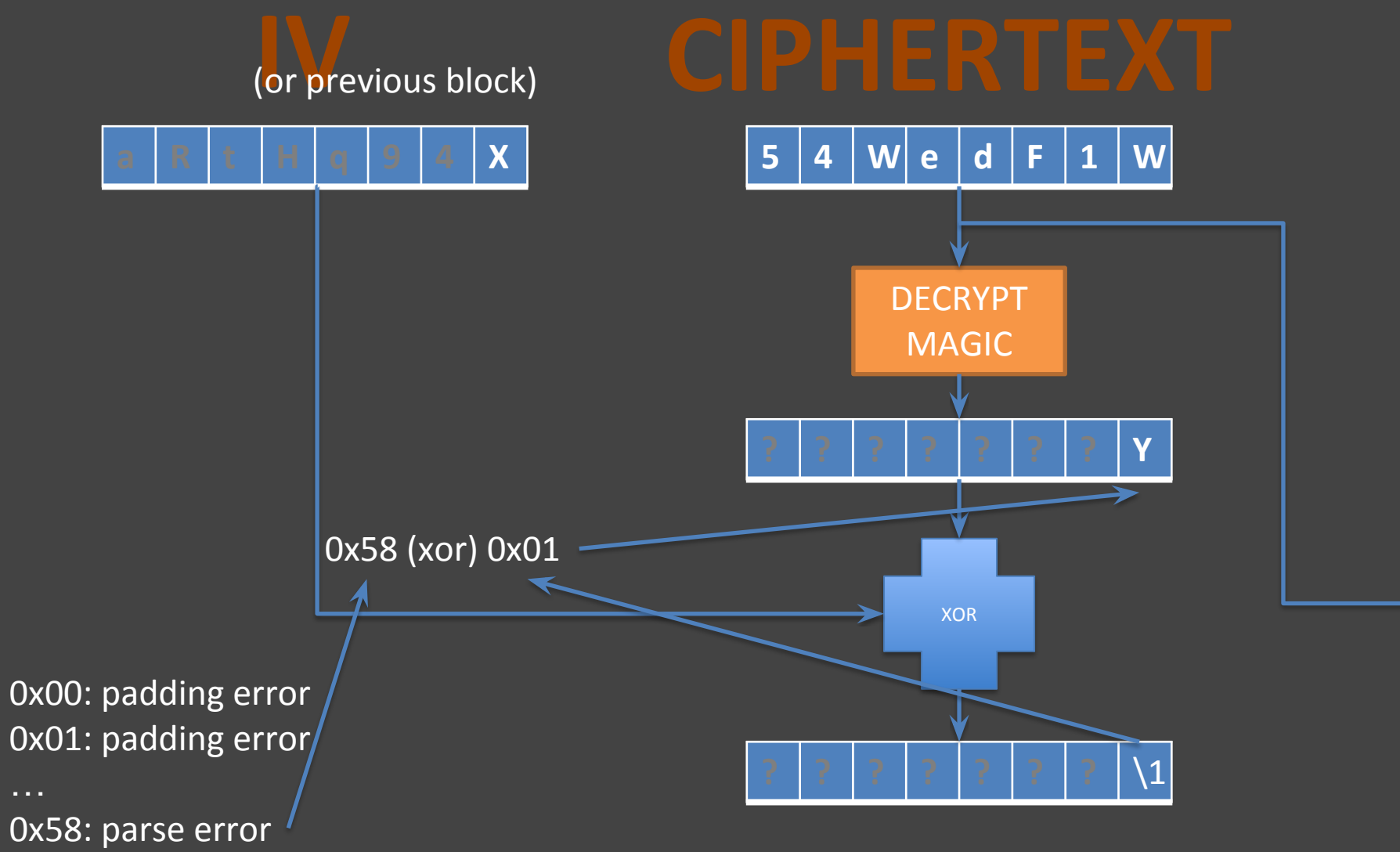
794ebcc5bd4e4e314d6447e4cf1fef504163e0b8f013990f81460d387d36ed9d781d3bd7aa290f0f

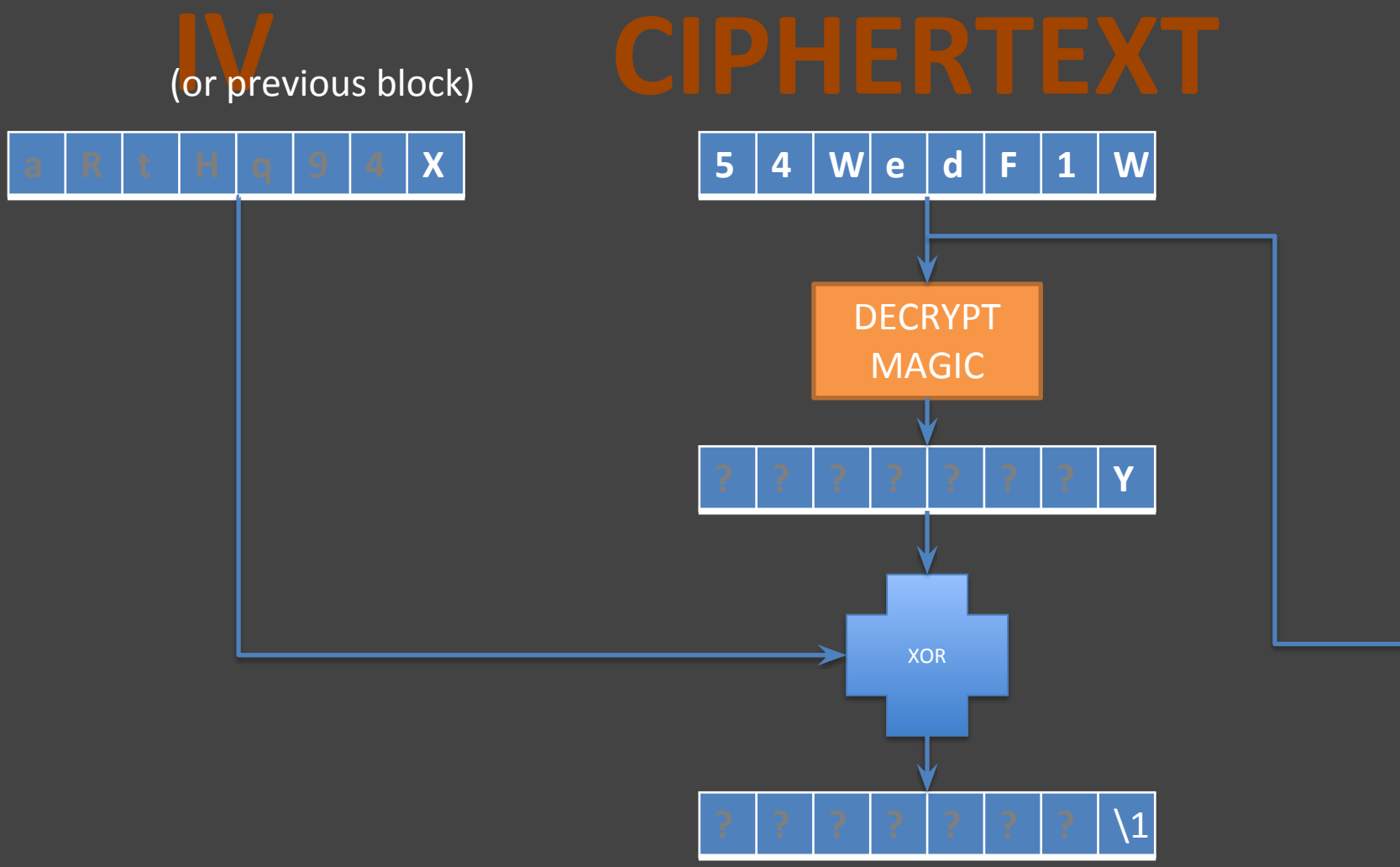
794ebcc5bd4e4e314d6447e4cf1fef504163e0b8f013990f81460d387d36
ed9d781d3bd7aa290f0f

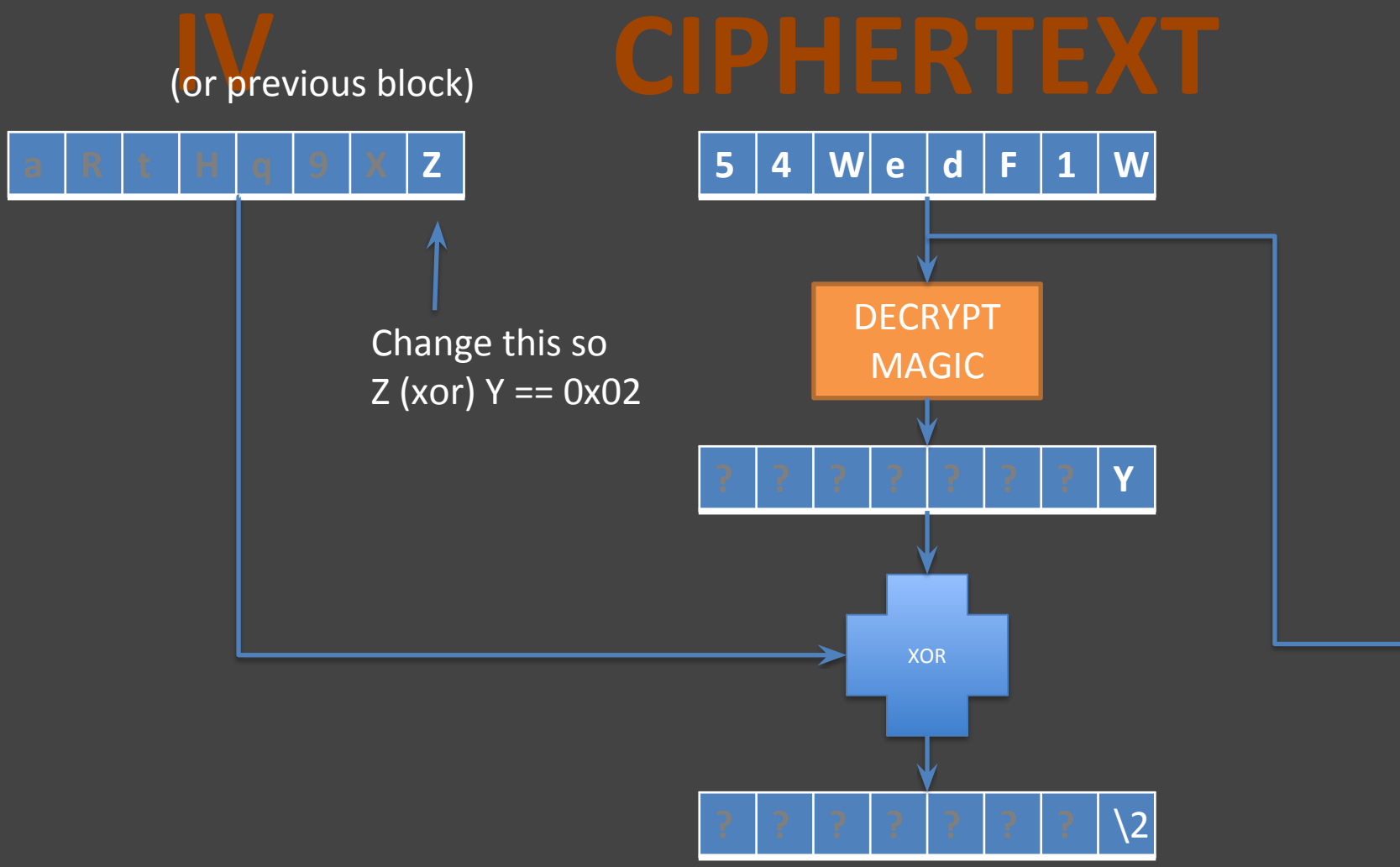


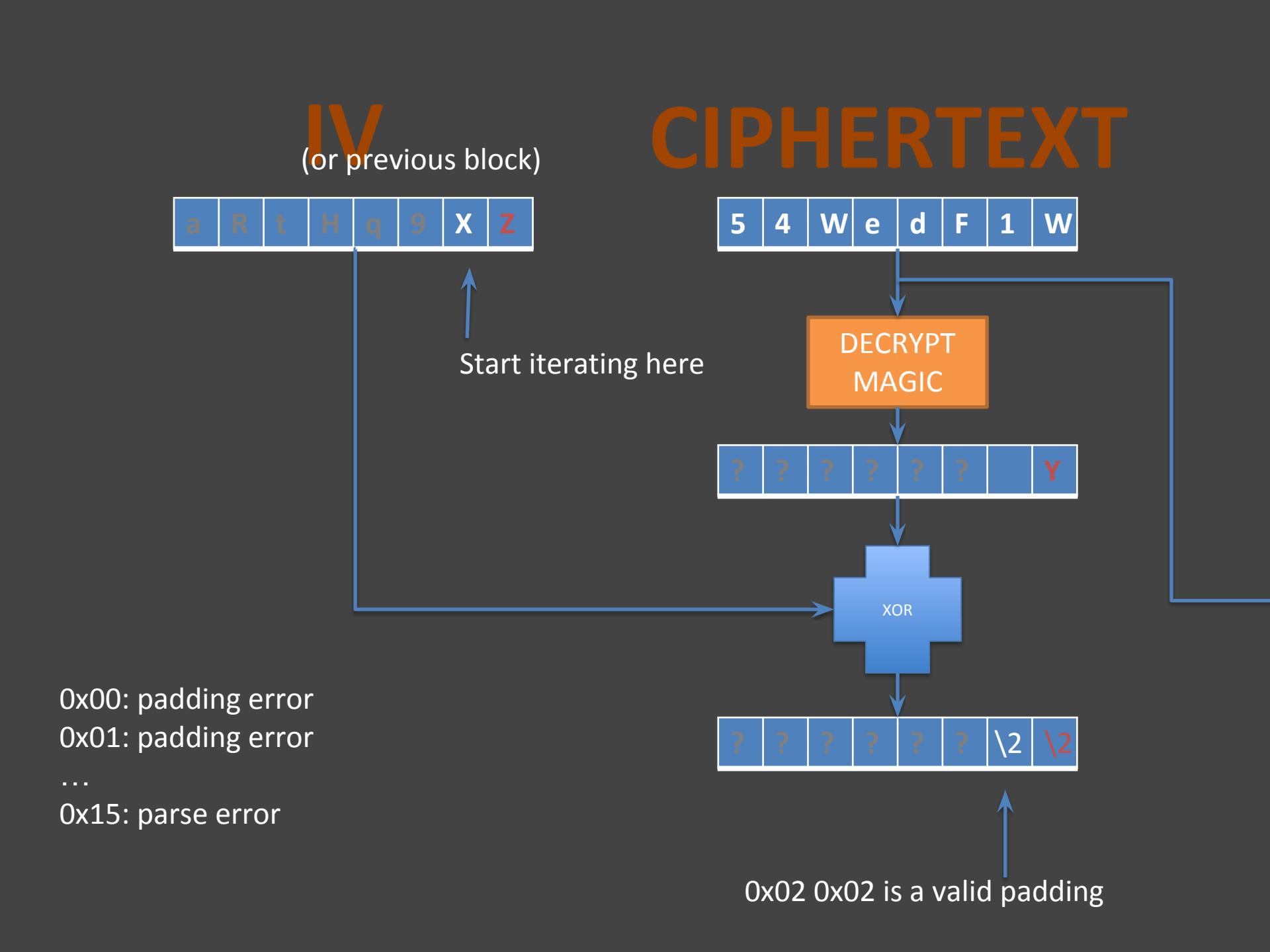

IV
(or previous block)
CIPHERTEXT
a R t H q 9 4 1
5 4 W e d F 1 W
DECRYPT MAGIC
F Q e W u I 6 7
XOR
{ “ a “ = 0 } 1

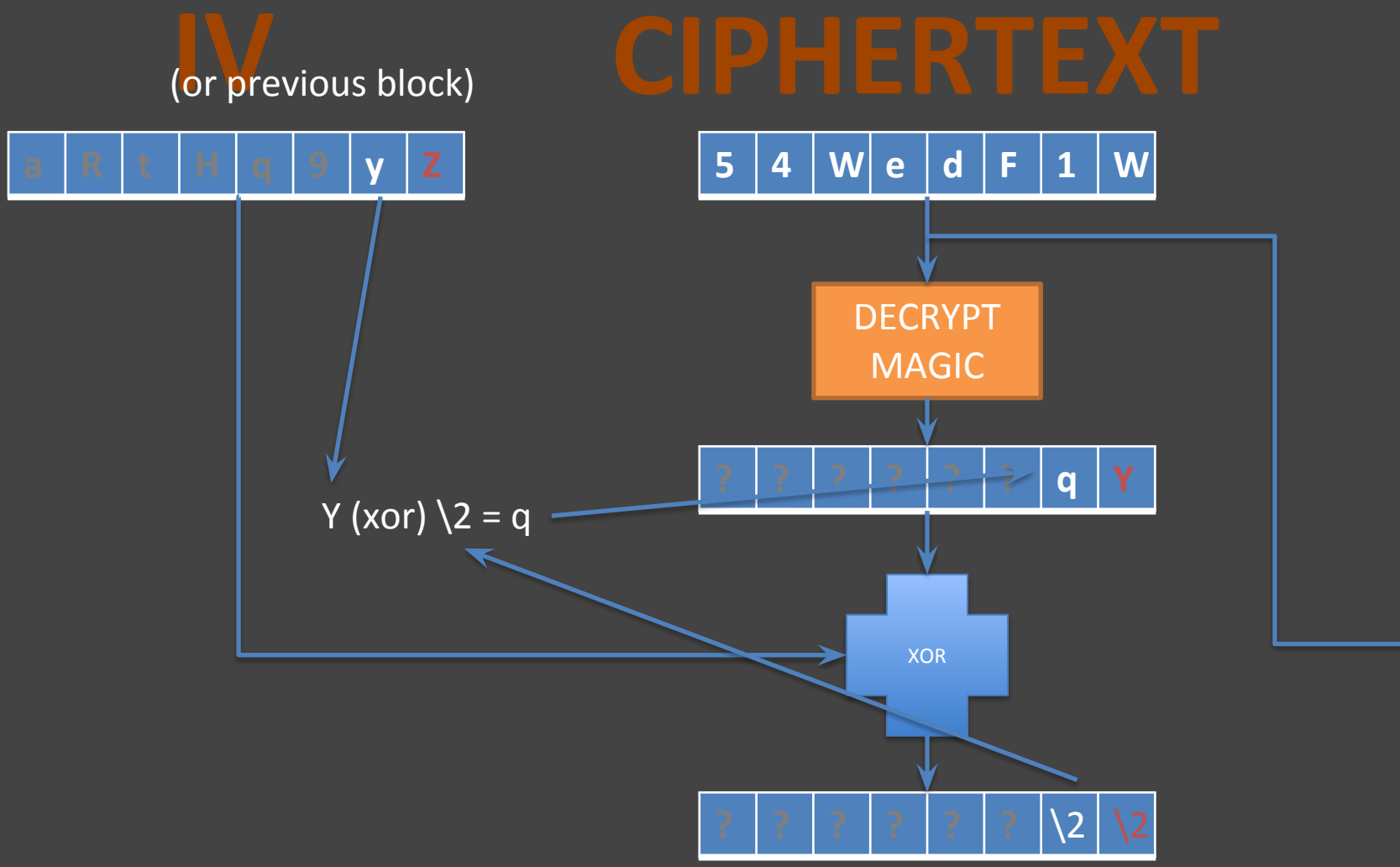

IV
(or previous block)
CIPHERTEXT
a R t H q 9 4 X
5 4 W e d F 1 W
Start changing this
DECRYPT MAGIC
? ? ? ? ? ? ?
XOR
? ? ? ? ? ? ? \1
0x00: padding error
0x01: padding error
…
0x58: parse error
0x01 is a valid padding

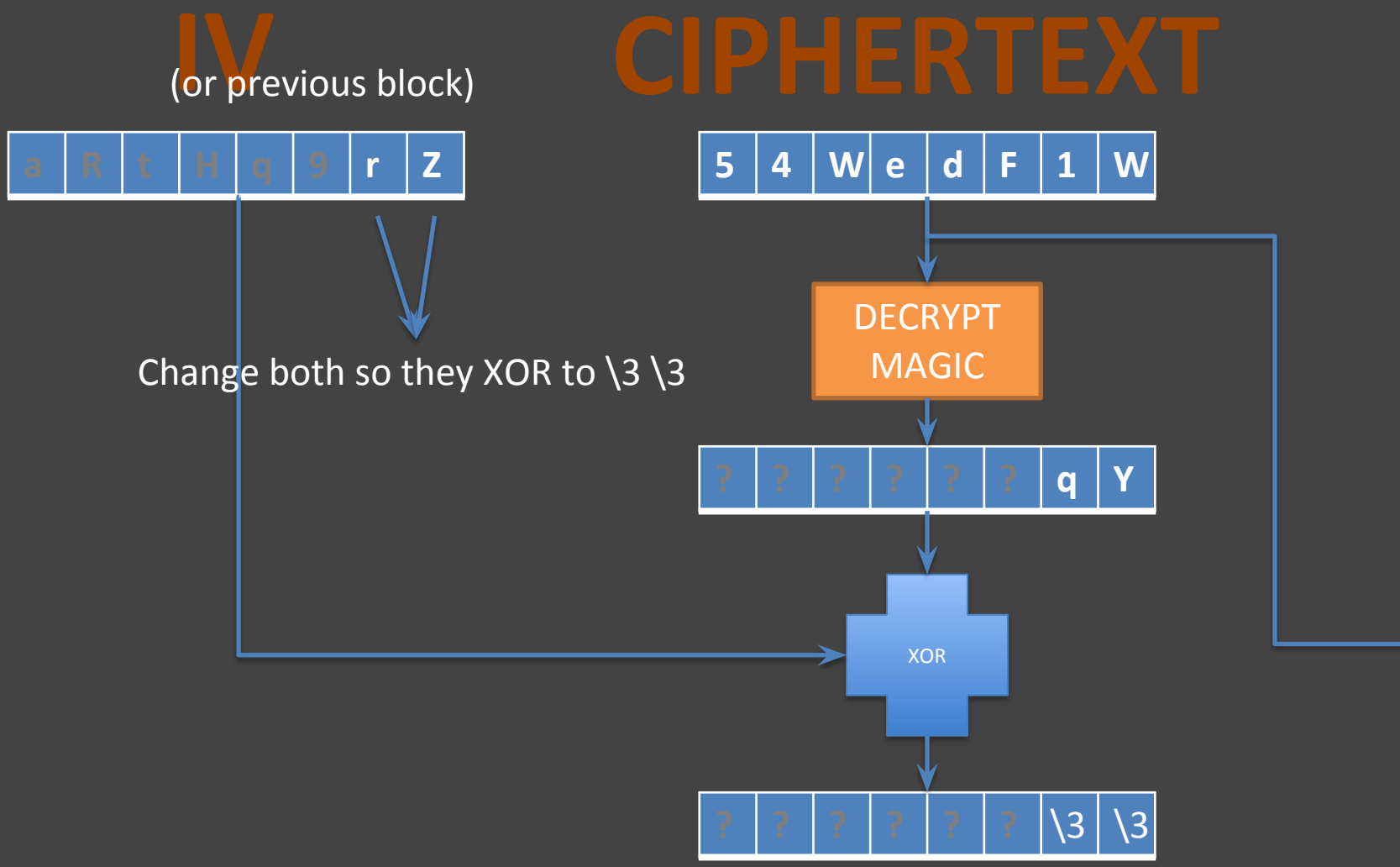

IV
(or previous block)
CIPHERTEXT
a R t H q 9 4 X
5 4 W e d F 1 W
DECRYPT MAGIC
? ? ? ? ? ? ? Y
0x58 (xor) 0x01
XOR
? ? ? ? ? ? ? \1
0x00: padding error
0x01: padding error
…
0x58: parse error


IV
(or previous block)
CIPHERTEXT
a R t H q 9 4 X
5 4 W e d F 1 W
DECRYPT MAGIC
? ? ? ? ? ? ? Y
XOR
? ? ? ? ? ? ? \1

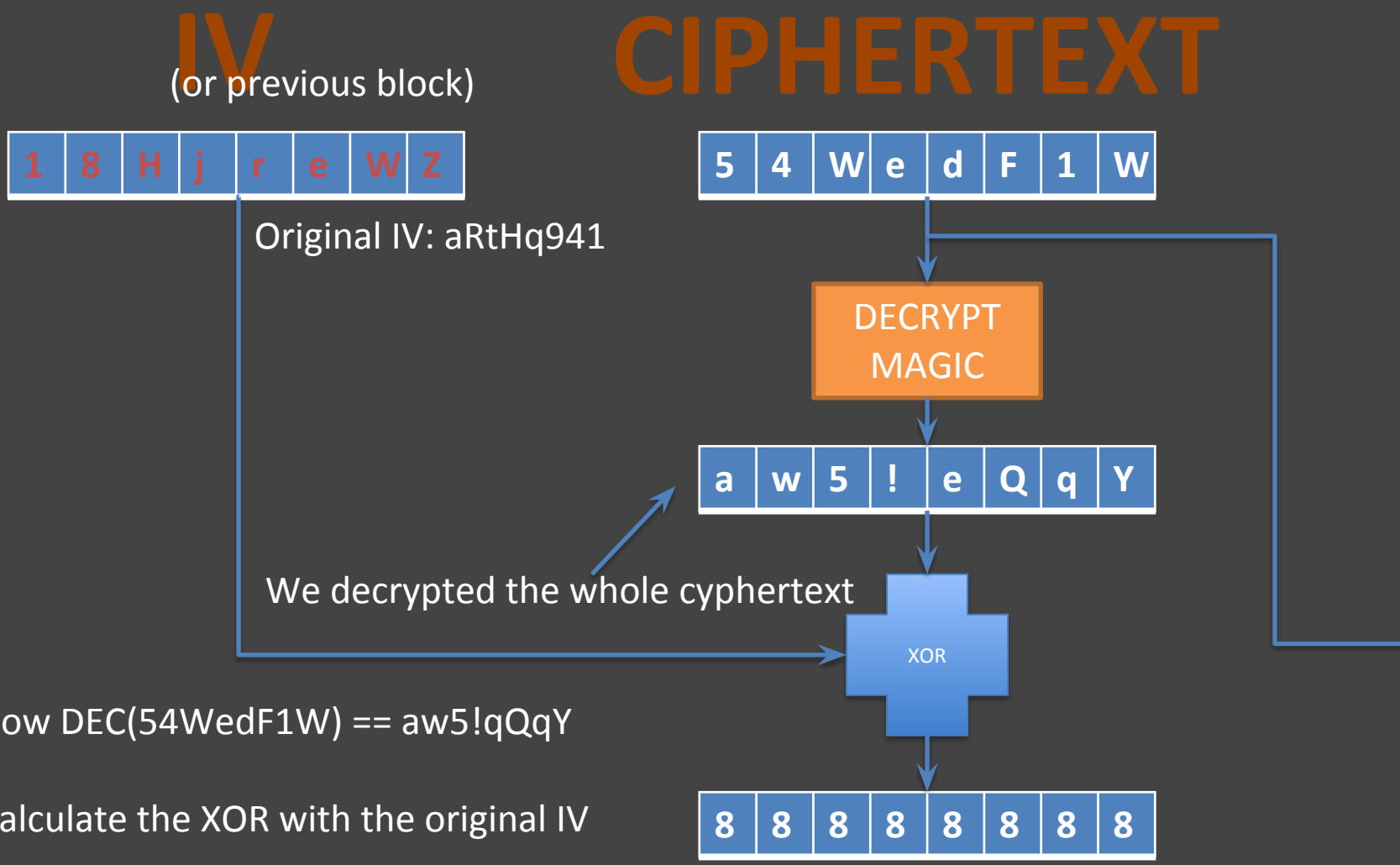

IV
(or previous block)
CIPHERTEXT
a R t H q 9 X Z
5 4 W e d F 1 W
Change this so
Z (xor) Y == 0x02
DECRYPT
MAGIC
? ? ? ? ? ? ? Y
XOR
? ? ? ? ? ? ? \2


IV
(or previous block)
CIPHERTEXT
a R t H q 9 X Z
5 4 W e d F 1 W
Start iterating here
DECRYPT MAGIC
? ? ? ? ? ? Y
XOR
? ? ? ? ? ? \2 \2
0x00: padding error
0x01: padding error
…
0x15: parse error
0x02 0x02 is a valid padding


IV
(or previous block)
CIPHERTEXT
a R t H q 9 y Z
5 4 W e d F 1 W
DECRYPT MAGIC
? ? ? ? ? ? q Y
Y (xor) \2 = q
XOR
? ? ? ? ? ? \2 \2


IV
(or previous block)
CIPHERTEXT
a R t H q 9 r Z
5 4 W e d F 1 W
Change both so they XOR to \3 \3
DECRYPT MAGIC
? ? ? ? ? ? q Y
XOR
? ? ? ? ? ? \3 \3

Do this until we arrive at padding \8 \8 \8 \8 \8 \8 \8 \8

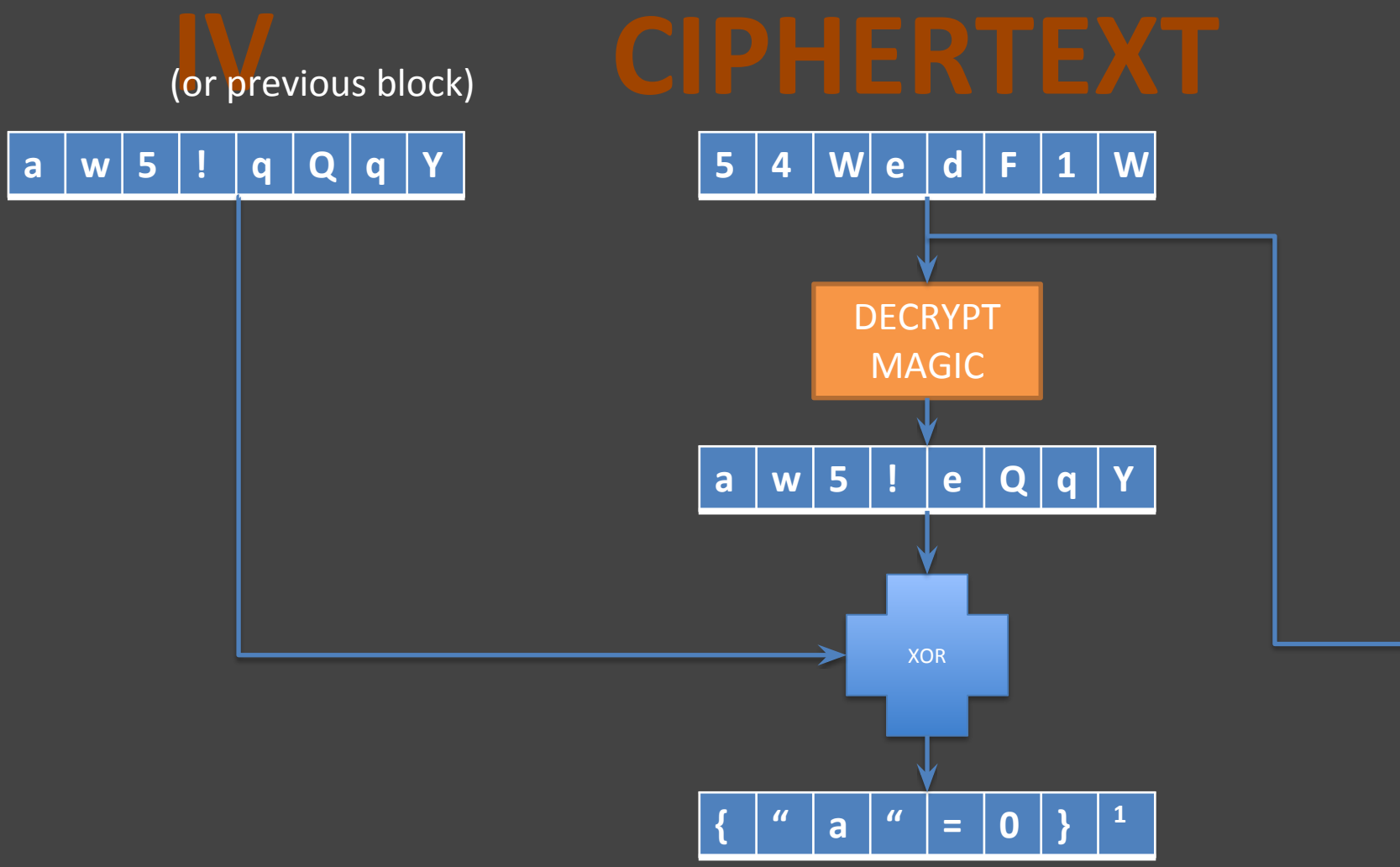


We know DEC(54WedF1W) == aw5!qQqY

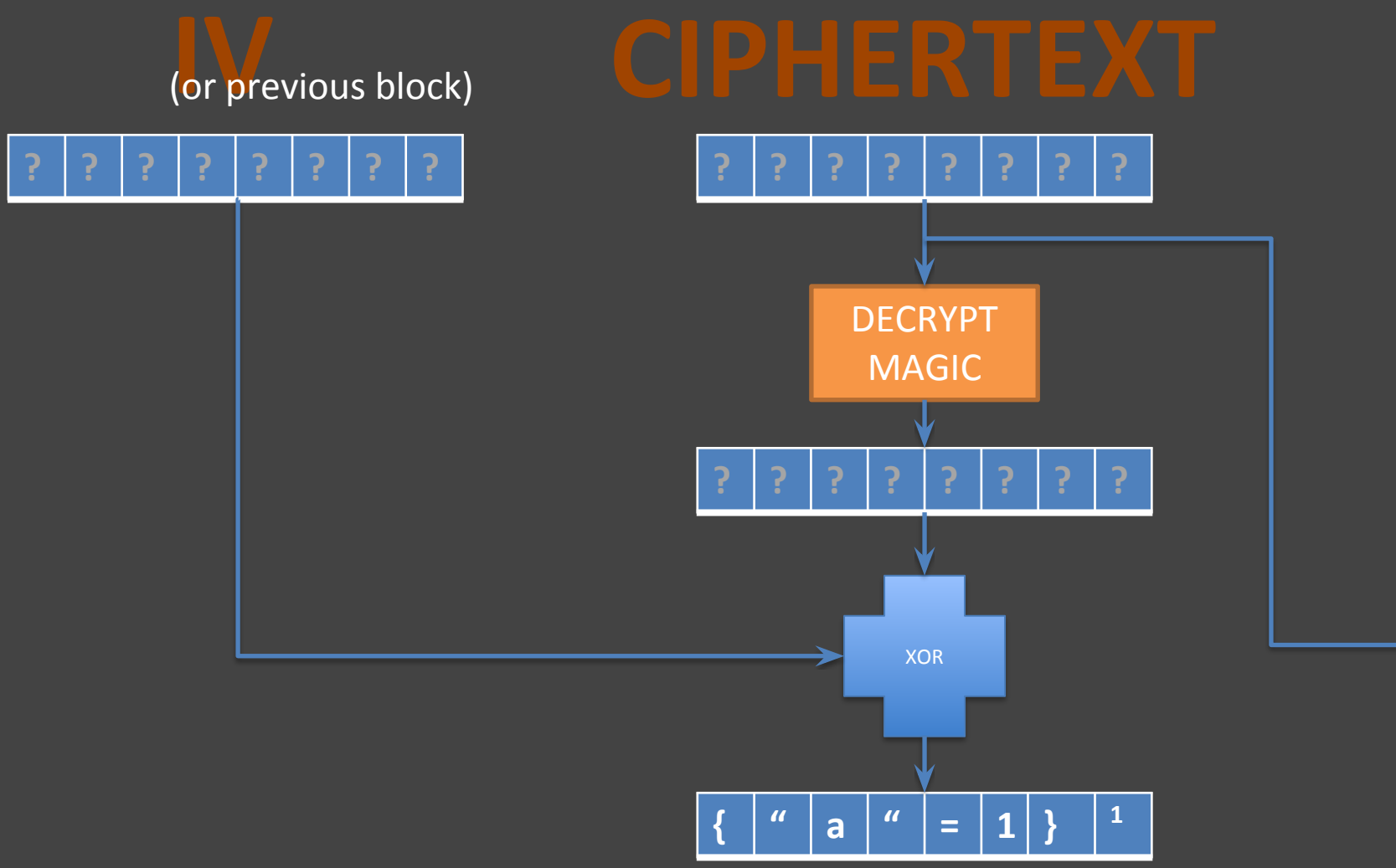
Let's calculate the XOR with the original IV

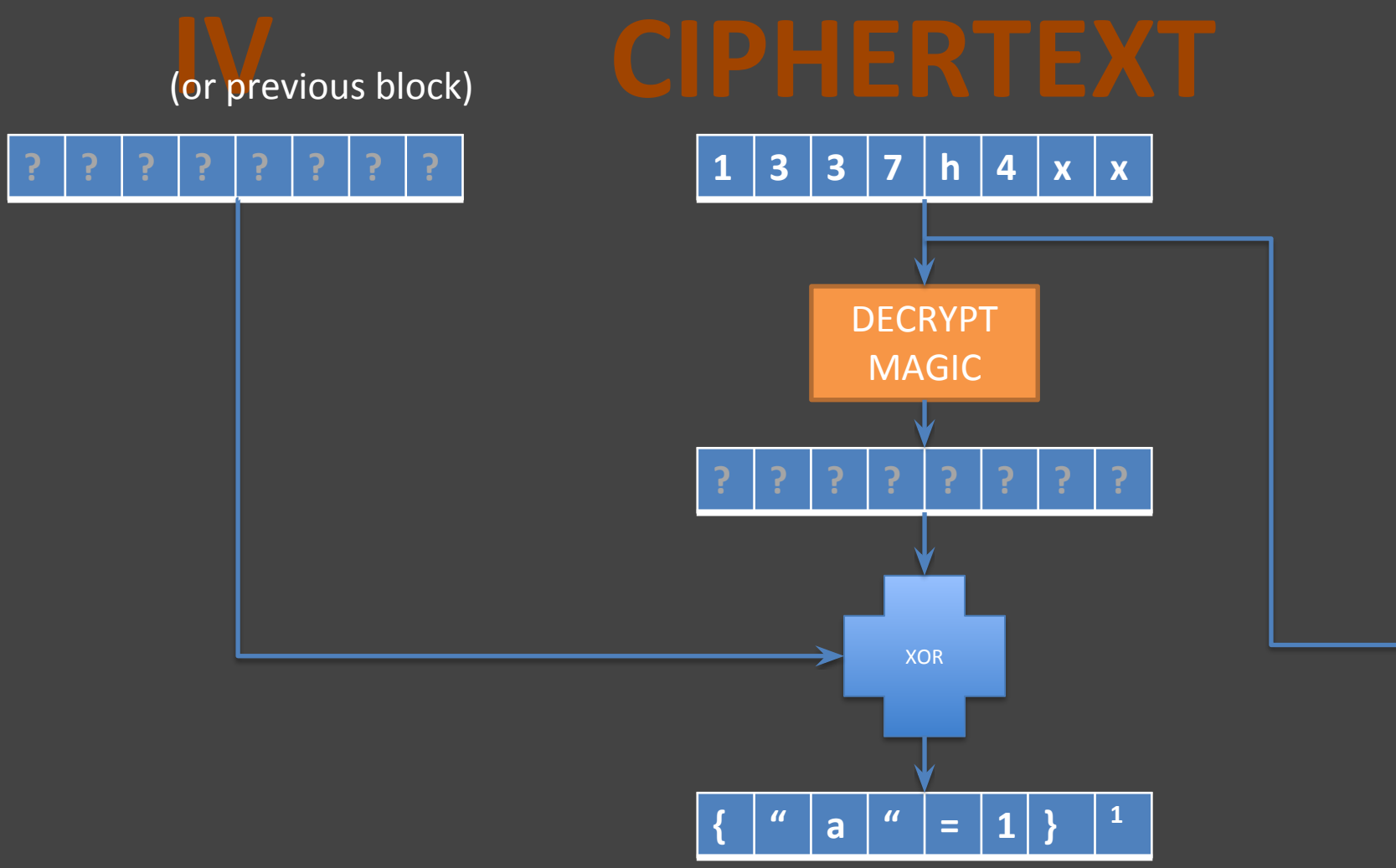
aw5!qQqY (XOR) aRtHq941 □ {"a"=0}

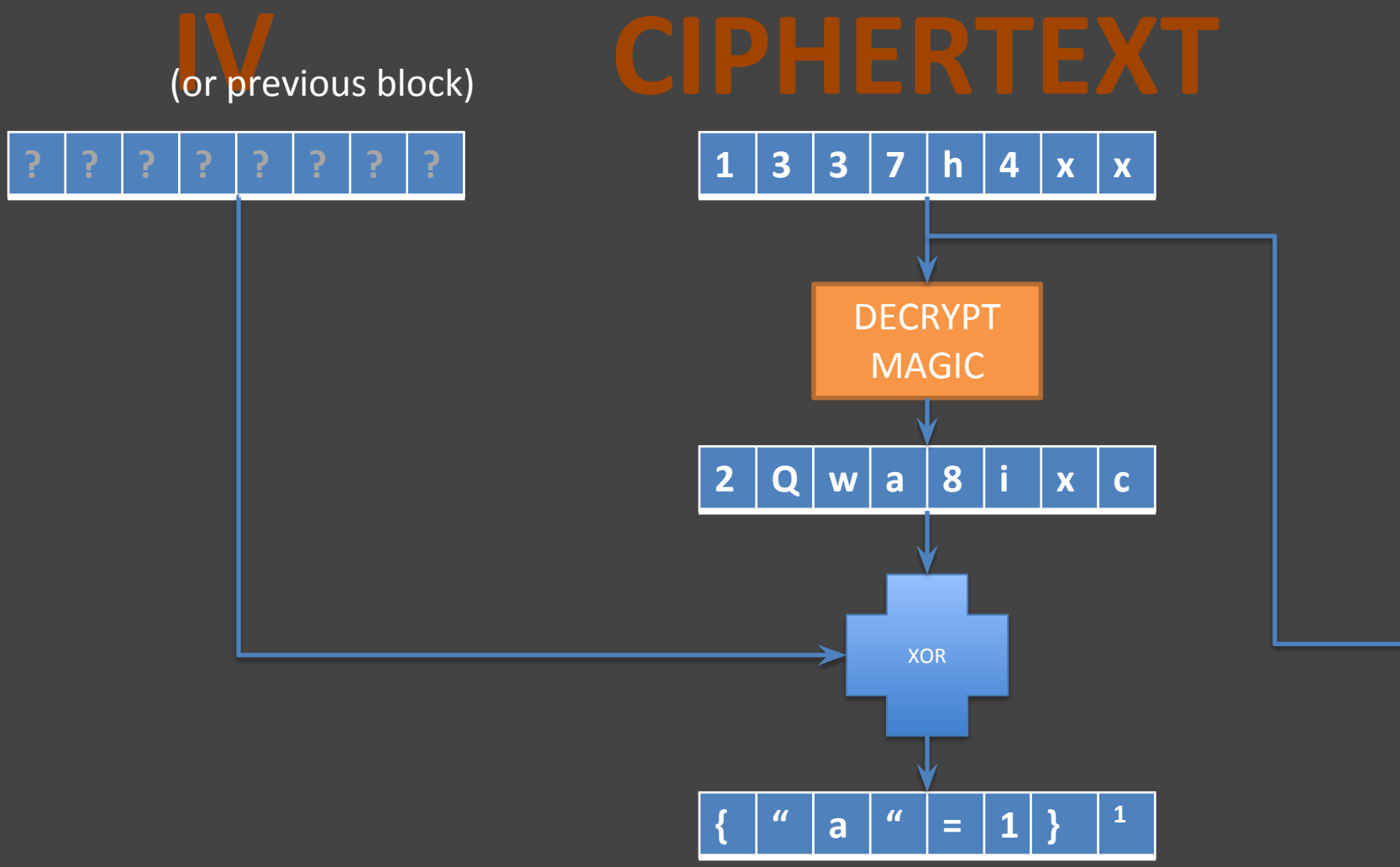
Using at most 8*256 = 2048 operations
(brute forcing would cost 256^8 = 18446744073709551615 operations)

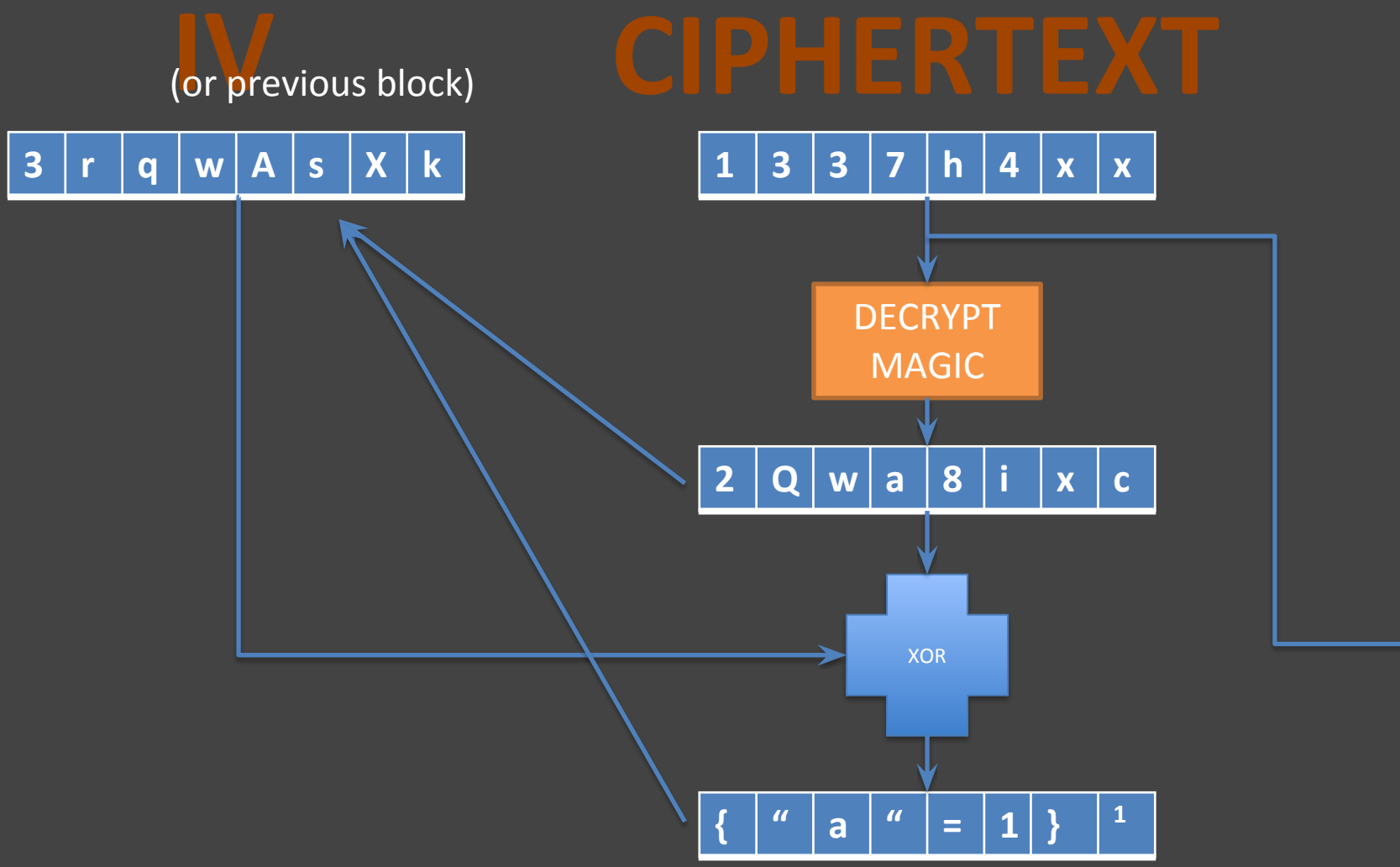

IV
(or previous block)
CIPHERTEXT
a w 5 ! q Q q Y
5 4 W e d F 1 W
DECRYPT MAGIC
a w 5 ! e Q q Y
XOR
{ “ a “ = 0 } 1

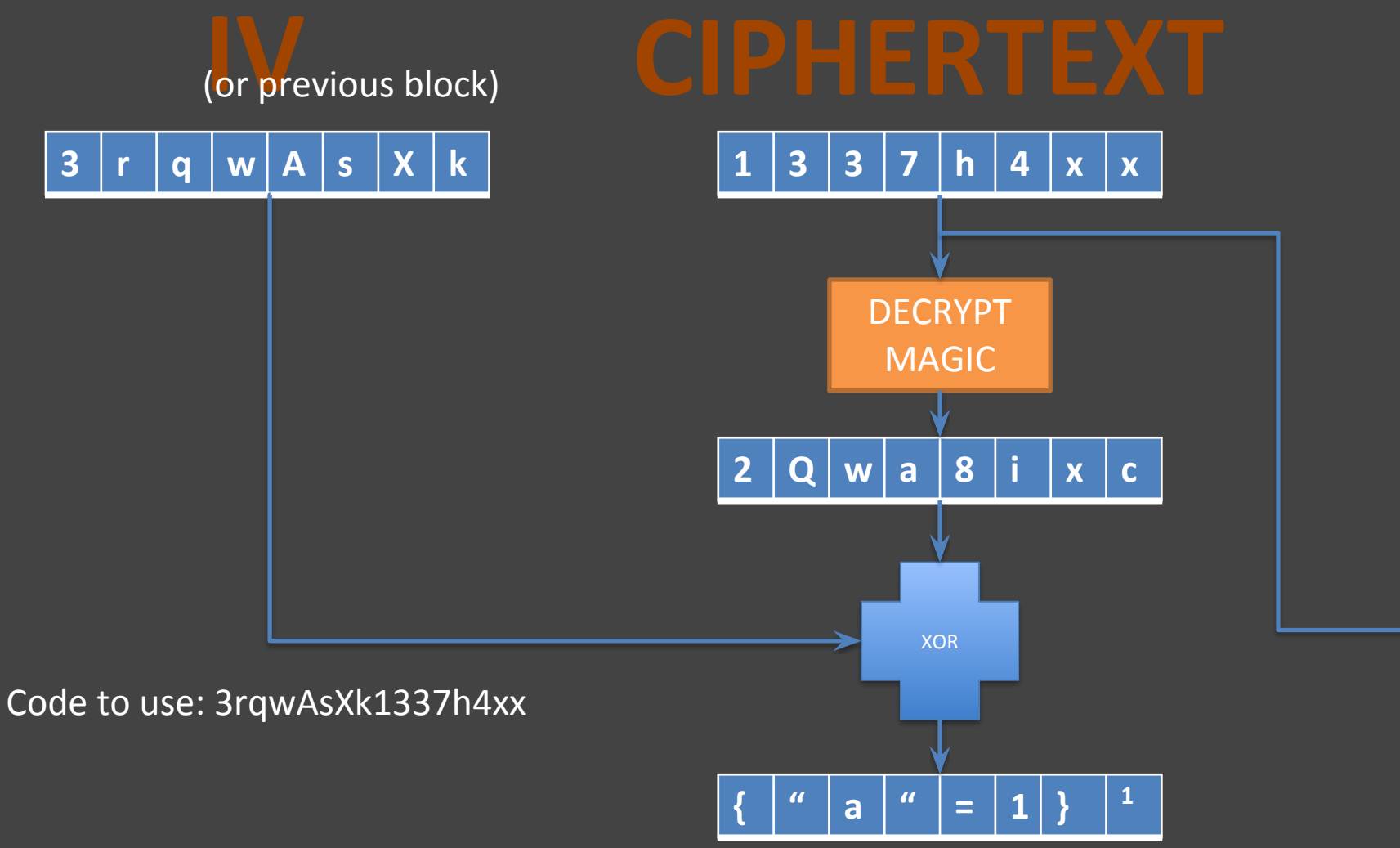
Are we done yet?


IV
(or previous block)
CIPHERTEXT
? ? ? ? ? ? ? ?
? ? ? ? ? ? ? ?
DECRYPT MAGIC
? ? ? ? ? ? ? ?
XOR
{ " a " = 1 } 1


IV
(or previous block)
CIPHERTEXT
? ? ? ? ? ? ? ?
1 3 3 7 h 4 x x
DECRYPT MAGIC
? ? ? ? ? ? ? ?
XOR
{ “ a “ = 1 } 1


IV
(or previous block)
CIPHERTEXT
? ? ? ? ? ? ? ?
1 3 3 7 h 4 x x
DECRYPT MAGIC
2 Q w a 8 i x c
XOR
{ “ a “ = 1 } 1


IV
(or previous block)
CIPHERTEXT
3 r q w A s X k
1 3 3 7 h 4 x x
DECRYPT MAGIC
2 Q w a 8 i x c
XOR
{ " a " = 1 } 1



Code to use: 3rqwAsXk1337h4xx

Are we done yet?

How to fix the following code?

```
BEGIN
    TRY: dec = DEC(input)
    CATCH: return “Input problem”
    END

    TRY: nopad = REMOVE_PAD(dec)
    CATCH: return “Padding problem”
    END

    TRY: data = PARSE(nopad)
    CATCH: return “Parse problem”
    END

    return “OKAY”
END
```

Like this?

```
BEGIN
        TRY: dec = DEC(input)
        CATCH: return “FAIL”
        END

        TRY: nopad = REMOVE_PAD(dec)
        CATCH: return “FAIL”
        END

        TRY: data = PARSE(nopad)
        CATCH: return “FAIL”
        END

        return “OKAY”
END
```

NO

Why?

```
BEGIN
    TRY: dec = DEC(input)
    CATCH: return “FAIL”
    END

    TRY: nopad = REMOVE_PAD(dec)
    CATCH: return “FAIL”
    END

    TRY: data = PARSE(nopad)
    CATCH: return “FAIL”
    END

    return “OKAY”
END
```

100ms

100ms

100ms

Or like this?

```
BEGIN
        TRY: dec = DEC(input)
        CATCH: sleep(random(200)); return “FAIL”
        END

        TRY: nopad = REMOVE_PAD(dec)
        CATCH: sleep(random(100)); return “FAIL”
        END

        TRY: data = PARSE(nopad)
        CATCH: return “FAIL”
        END

        return “OKAY”
END
```

Implementation note: Canvel et al. [CBCTIME] have demonstrated a timing attack on CBC padding based on the time required to compute the MAC. In order to defend against this attack, implementations MUST ensure that record processing time is essentially the same whether or not the padding is correct.

Source: RFC5246 (TLS 1.2)

Or like this?

Data is now:
<data><padding><MAC>

```
BEGIN
        TRY: dec = DEC(input)
        CATCH: return “FAIL”
        END

        TRY: maccheck(dec)
        CATCH: return “FAIL”
        END

        TRY: nopad = REMOVE_PAD(dec)
        CATCH: return “FAIL”
        END

        TRY: data = PARSE(nopad)
        CATCH: return “FAIL”
        END

        return “OKAY”
END
```

We are done!