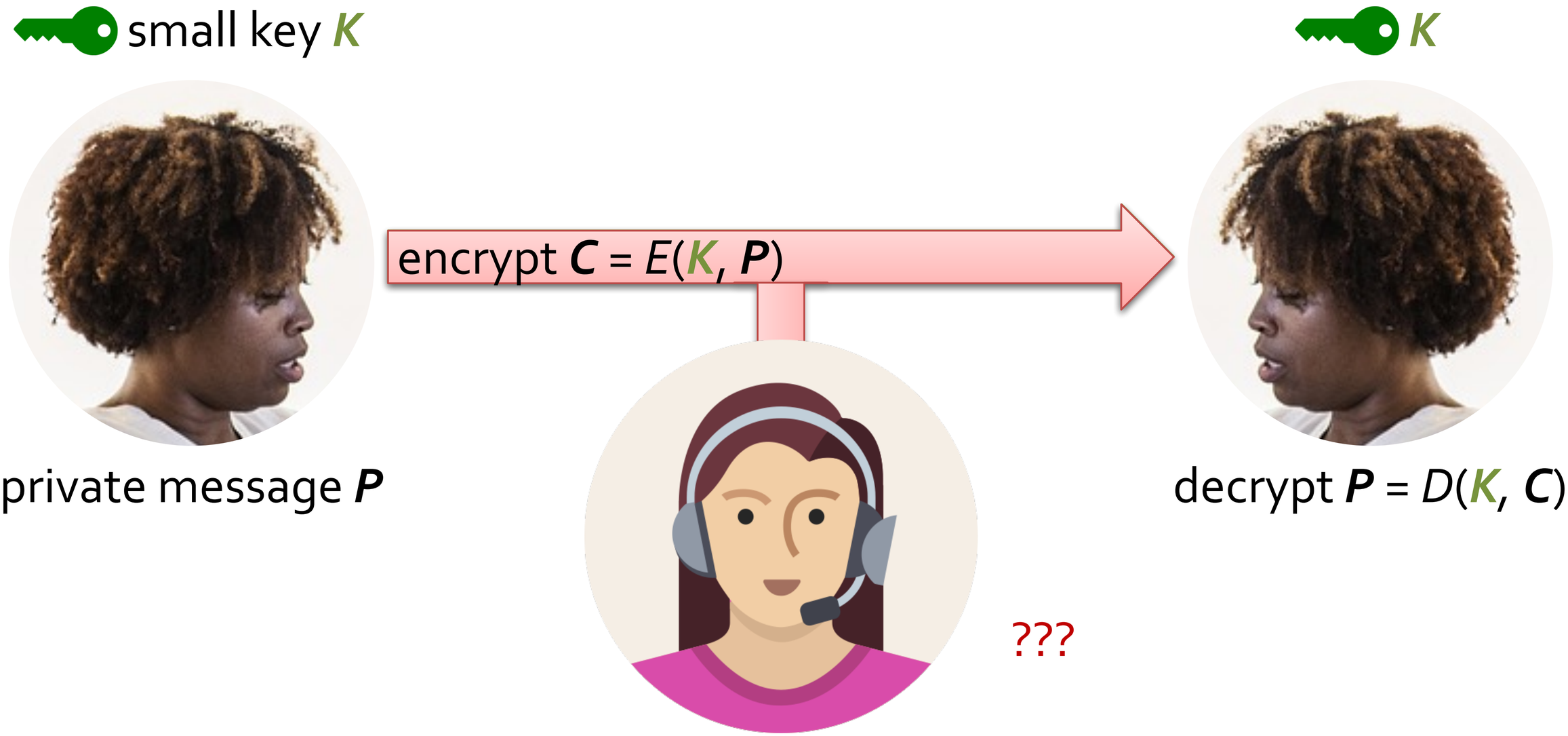
Lecture 4: Encryption via enciphering

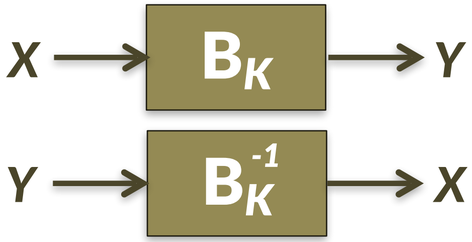
- Homework 2 due Monday 2/3
- Textbook reading for this week: *Serious Cryptography*, chapter 4
- I will try to post lecture slides before class (no promises though)

Recap: protecting data confidentiality at rest



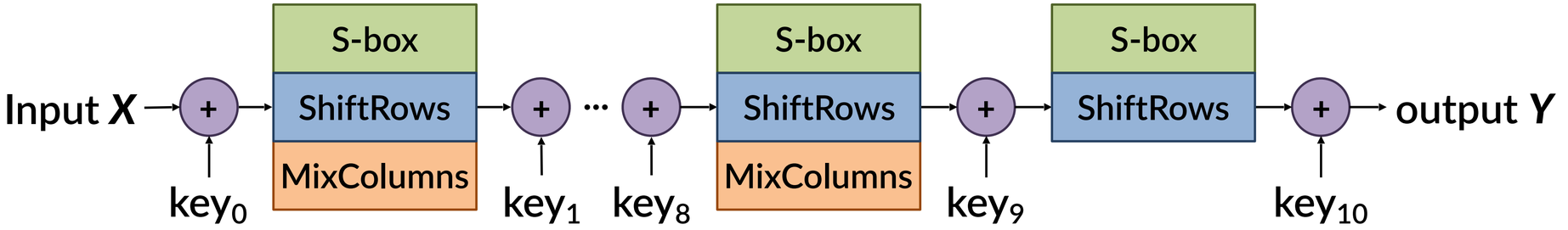
Recap: block cipher

- Family of finite permutations (i.e., codebooks), indexed by a secret key
  - Forward direction called *enciphering*
  - Backward direction called *deciphering*



- B_K is *strongly pseudorandom* if every resource-bounded adversary can only distinguish the real cipher B_K from Π with very small probability ε

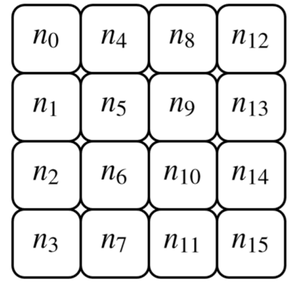
Recap: Rijndael, aka AES, is a concrete block cipher



Key alternating structure

- 128, 192, or 256 bit initial key
- Expand into $r+1$ round keys, each of which is 128 bits long
- Invertible key schedule: given key_i, can compute key_{i-1} or key_{i+1}

Iterated round structure



- 16 bytes of state
- Total of $r = 10$ to 14 rounds
- 3 invertible operations per round
  - Final round is slightly different
- Only S-box is nonlinear

Recap: brute force attack

- There is a large (but finite!) set of possible keys 
- *Brute force attack*: Eve runs a **for** loop over all keys, checks if $B_K(X) = Y$
- Ergo, best possible time bound t = 2^λ
- Two ways crypto can go bad
  - **Obsolete:** it is computationally feasible to run a brute force attack (e.g., DES)
  - **Broken:** there exists an attack that runs faster than brute force (e.g., 2DES)

Game	Search size	Solved?
Connect 4	2^43	✓
Limit hold 'em	2^47	✓
Checkers	2^67	✓
Modern crypto	2^128–2^256	
Chess	2^133	
No limit hold 'em	2^465	
Go (19 × 19)	2^568	

How to think about really large numbers

Converting to base-2

$2^{10} \approx 10^{3}$ (kilo)

$2^{20} \approx 10^{6}$ (mega)

$2^{30} \approx 10^{9}$ (giga)

$2^{40} \approx 10^{12}$ (tera)

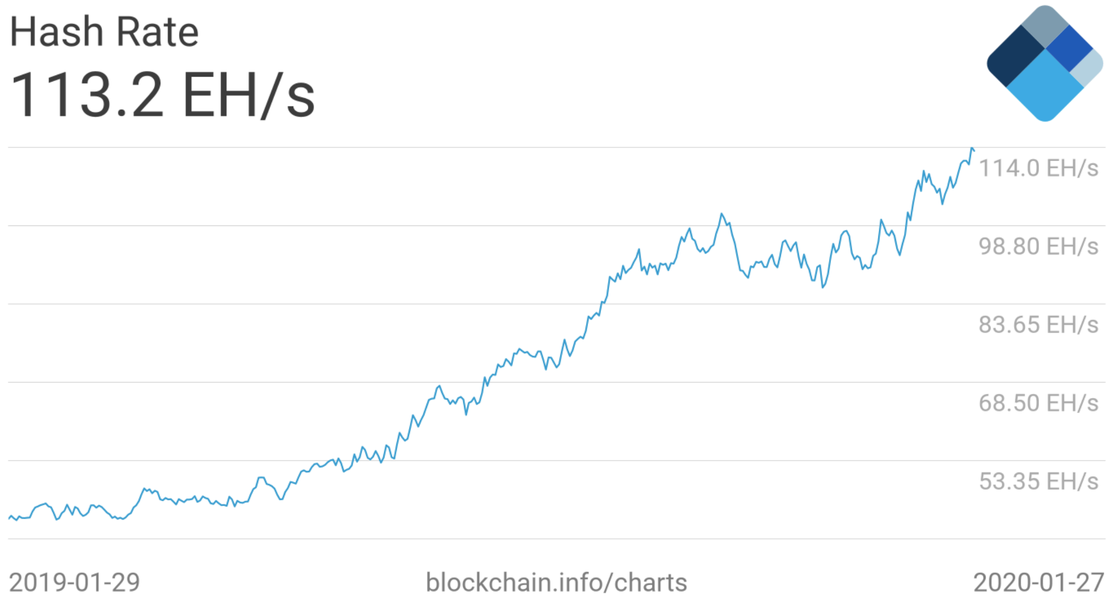
$2^{50} \approx 10^{15}$ (peta)

$2^{60} \approx 10^{18}$ (exa)

⋮

Speed of cryptography on modern computers

- Running AES: $(2^{31}\ {}^{\text{cycles}}/_{\text{sec}}) / (10\ {}^{\text{cycles}}/_{\text{aes}}) \approx 2^{27}\ {}^{\text{aes}}/_{\text{sec}}$
  - So about $2^{52}\ {}^{\text{ops}}/_{\text{year}}$, given that 1 year ≈ 2^{25} sec
- Entire bitcoin network: about $2^{67}\ {}^{\text{ops}}/_{\text{sec}}$



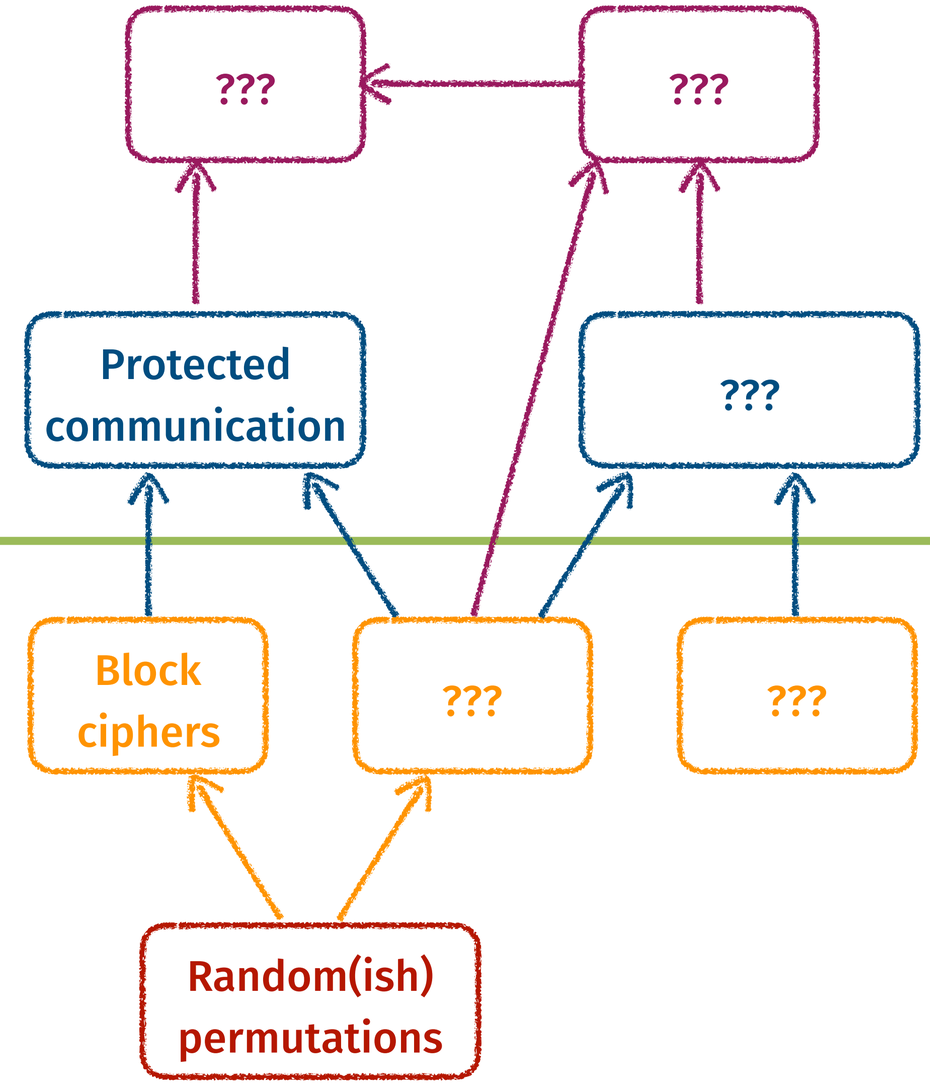
Difficulty of attacking crypto

Eve's search space	Time with laptop (2^{52} ops/yr)	Time with bitcoin network (2^{67} ops/sec)
2^{20} (your homework)	0.01 second	~instantaneous
2^{56} (DES brute force)	2^{4} = 16 CPU core-years	much less than 1 second
2^{80}	2^{28} = 256 million CPU core-yr	2^{13} seconds ≈ 2 hours
2^{128} (AES-128 brute force)	2^{76} = 64 sextillion CPU core-yr = 2^{33} x 2^{43} = 8 trillion CPU core-yr for each person on earth	2^{63} sec ≈ 2^{38} year ≈ 100 billion yr (cf. age of universe ≈ 14b years)
2^{256} (AES-256 brute force)	(about the energy of the sun)	

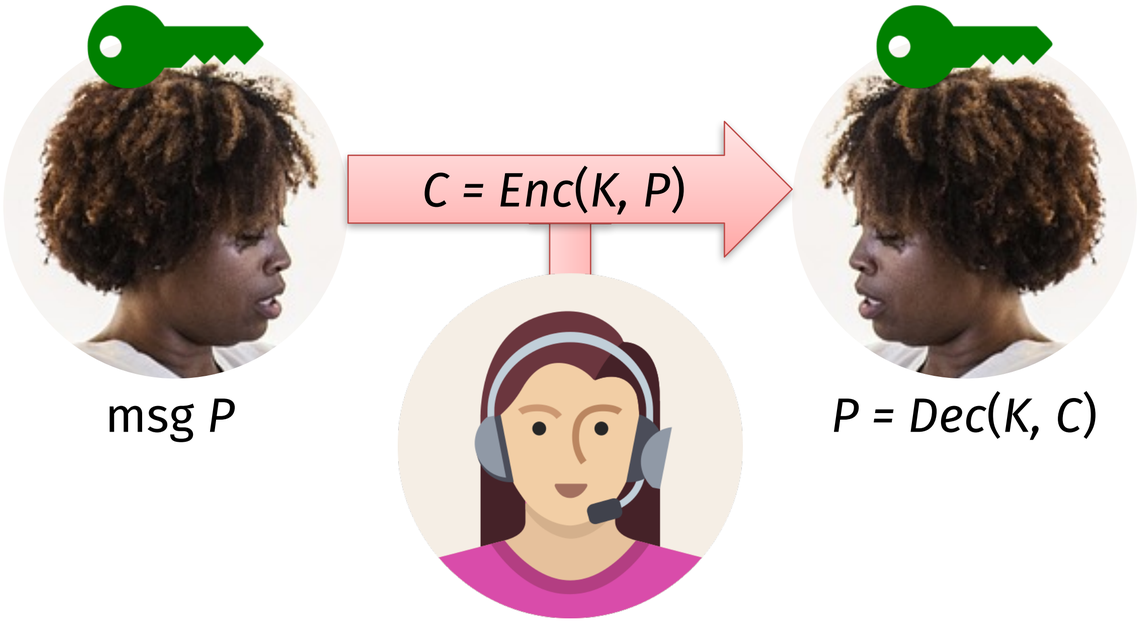
Crypto in this course

Elegant protocols

Utilitarian tools



Alice's confidentiality + integrity goals

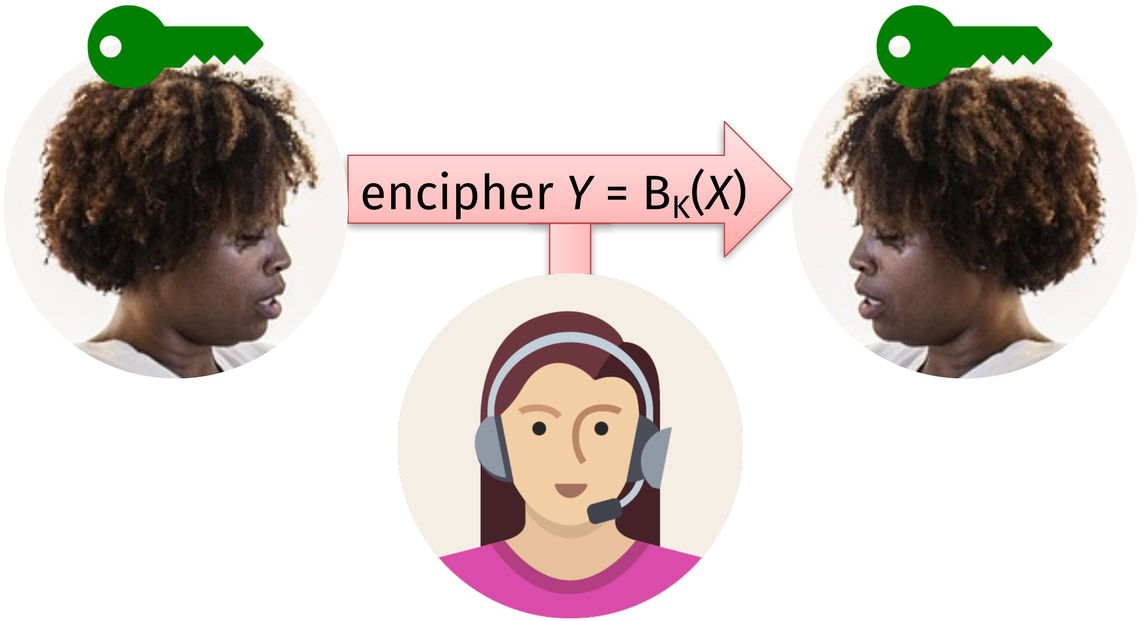


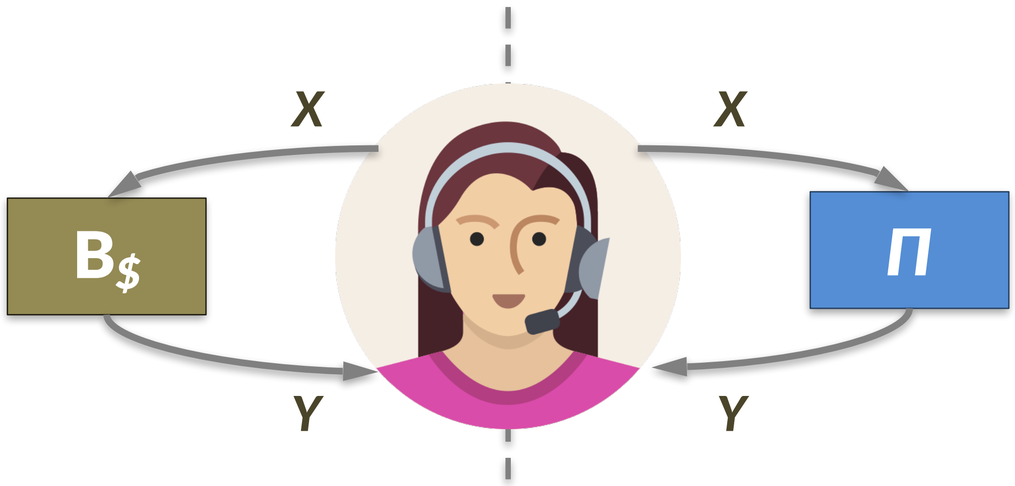
- ***Data privacy*: Eve cannot learn *P* (today)**
- *Data authenticity*:
  if Eve tampers with *C*, then Alice can detect the change
- *Entity authenticity*:
  future Alice knows that she previously created *C*

"Confidentiality xor authenticity is **not possible**.
If you don't have both, often you don't have either."

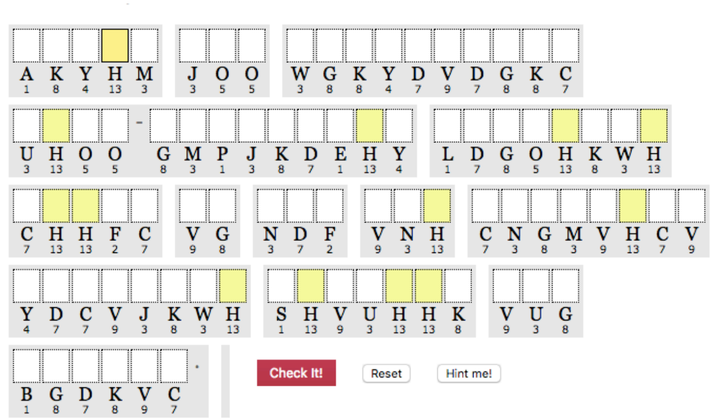
–Prof. Matthew Green, Johns Hopkins

Does a cipher give us confidential communication?





- First reaction: Yes! The cipher transforms *X* in a *confusing* way
- Full answer: not quite, we have 2 issues
  - Usability: Block cipher only supports messages where |*X*| = block length
  - Security: Vulnerable to *frequency attacks* if Alice enciphers the same block twice

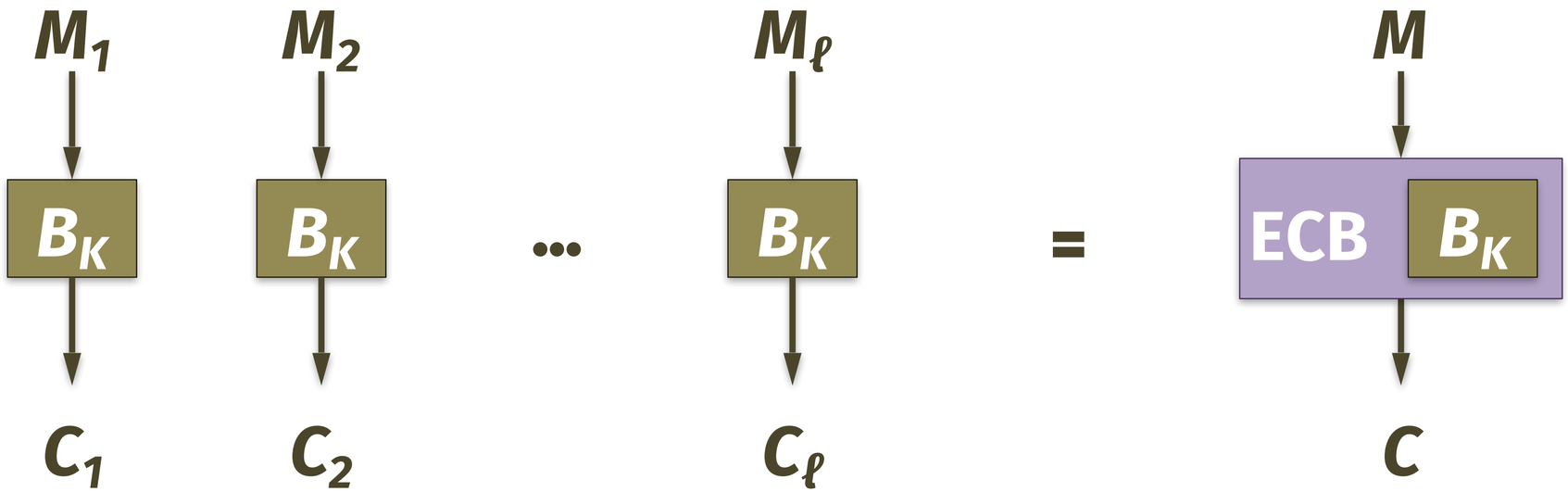


Supporting longer messages

Def. A ***mode of operation*** connects multiple calls to a block cipher (with one key K)

One simple mode: process each block of the message independently

This is called ***Electronic Codebook (ECB)*** mode



ECB mode ⇒ sad Linux penguins

Raw image of
Linux penguin

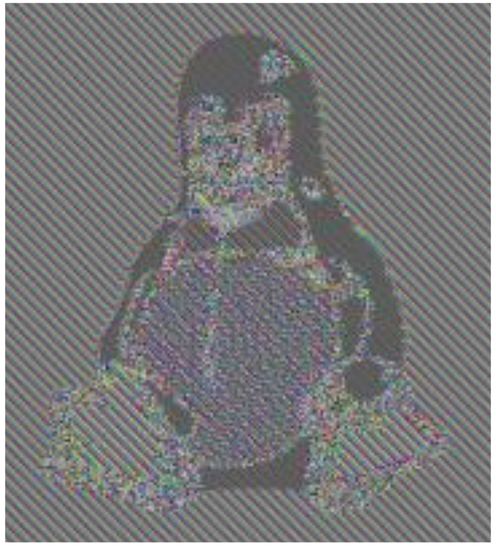
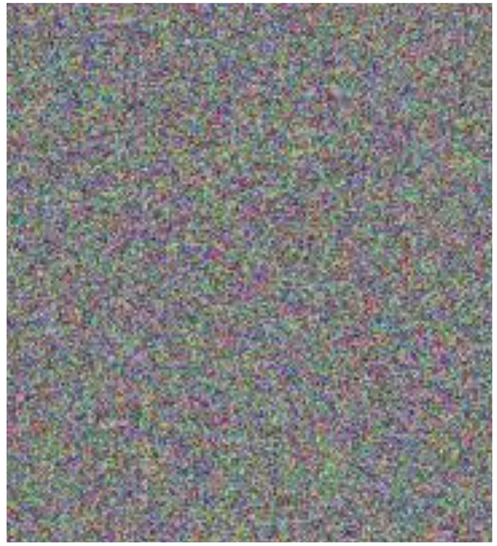


Image after
ECB mode



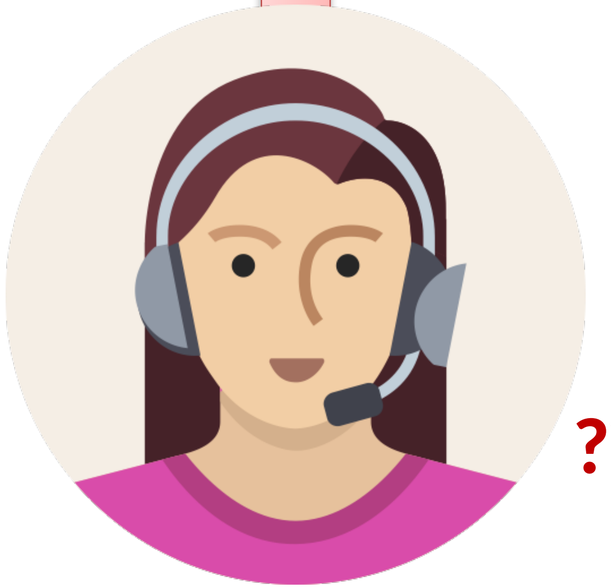
What we want
encryption to do

What if message blocks *don't* repeat?

key K

private data
$P_1, P_2, \ldots P_\ell$

encode $C_i = B_K(P_i)$



???

key K

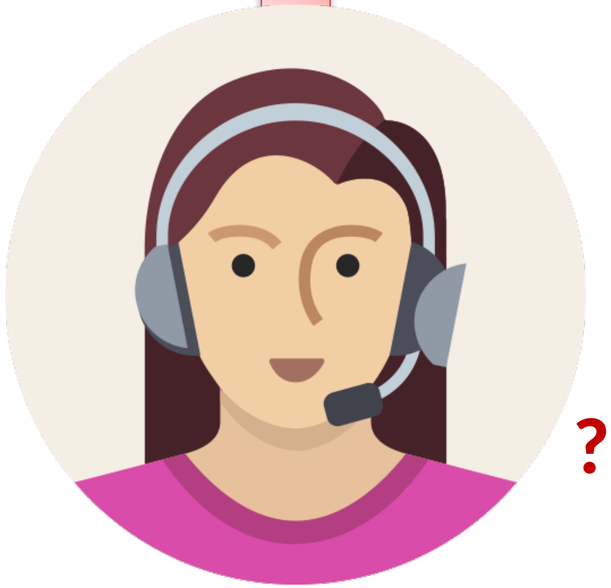
decode $P_i = B_K^{-1}(C_i)$

What if message blocks *don't* repeat?

key K

private data
$P_1, P_2, \ldots P_\ell$

encode $C_i = \Pi(P_i)$



???

key K

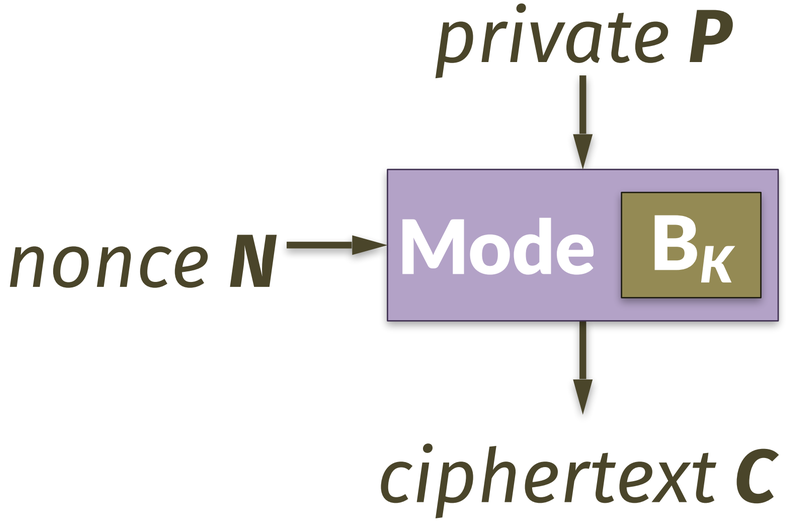
decode $P_i = \Pi^{-1}(C_i)$

How do we guarantee that message blocks don't repeat?

Lessons learned

- **Randomness matters:** We can confuse Eve! Just need to design a mode of operation that guarantees each enciphered block is unique.

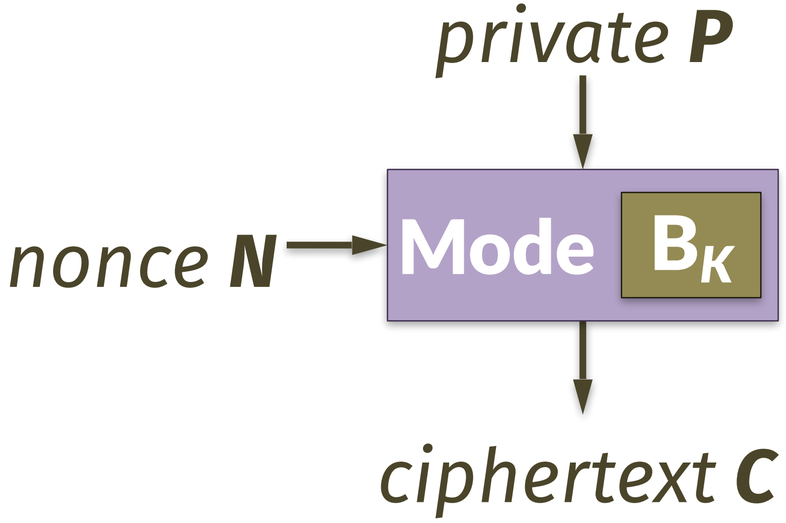
- **Definitions matter:** Argument leverages the concept that a block cipher “looks like” a random permutation from Eve’s point of view.



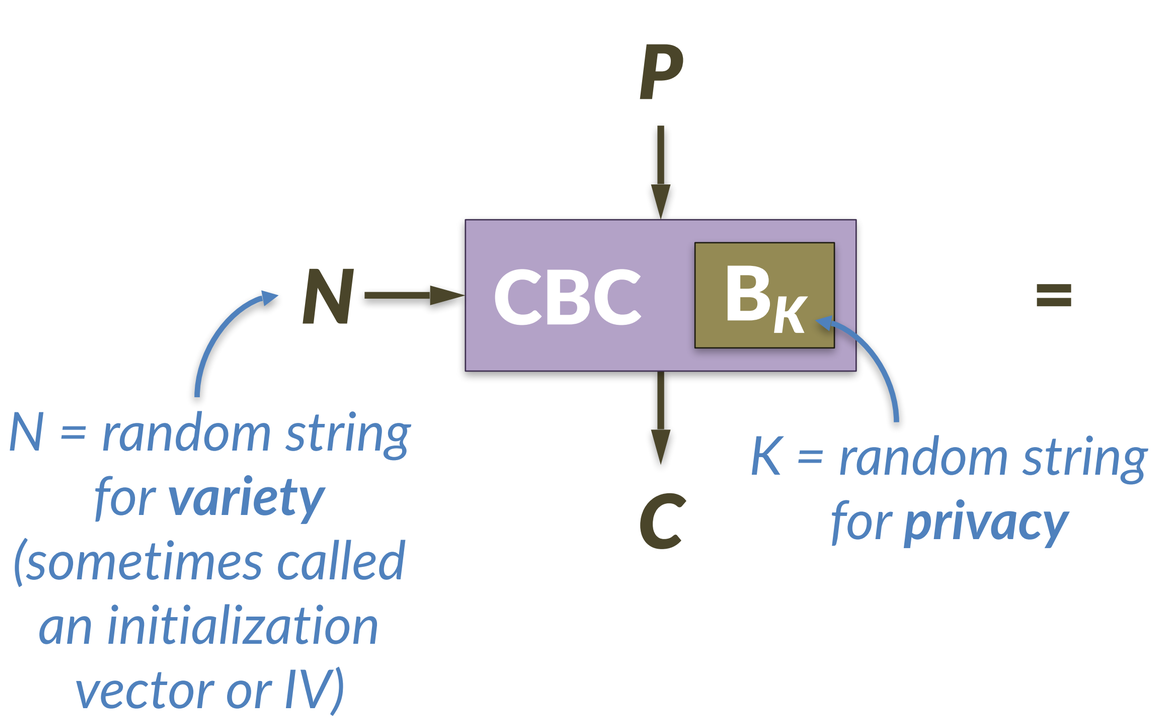
Lessons learned

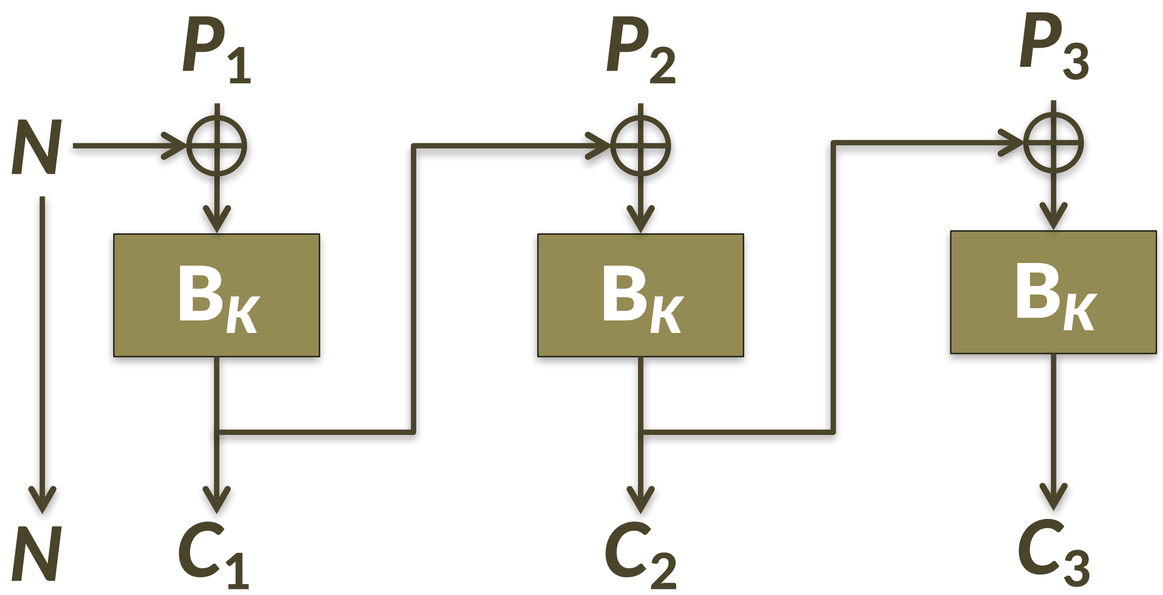
- **Randomness matters:** We can confuse Eve! Just need to design a mode of operation that guarantees each enciphered block is unique.

- **Definitions matter:** Argument leverages the concept that a block cipher “looks like” a random permutation from Eve’s point of view.

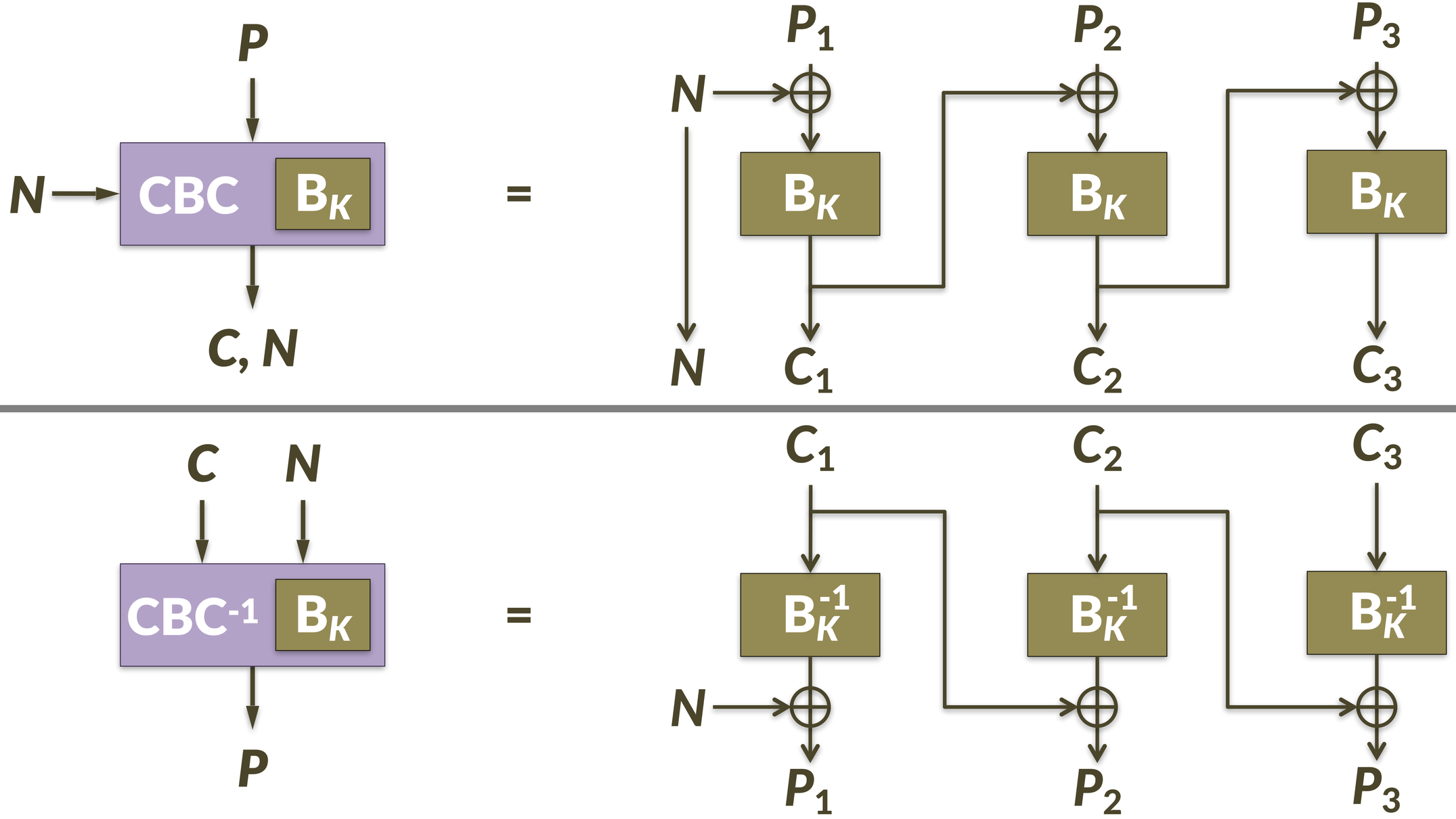


Cipher block chaining (CBC) mode





CBC decryption



Apple's Common Crypto Library Defaults to a Zero IV if One is not Provided

Today I was writing some guidelines about generating keys for mobile applications at work. While providing code examples in Java and Obj-C for AES encryption I happened to look at Apple's Common Crypto library . While going through the source code for CommonCryptor.c, I noticed that IV is commented as `/* optional initialization vector */`. This makes sense because not all ciphers use IV and not all AES modes of operation (e.g. ECB mode). However; if an IV is not provided, the library will default to a zero IV.

You can see the code here inside the function `ccInitCryptor` (search for defaultIV) source. `CC_XZEROMEM` resets all bytes of IV to zero (that is 0x00):

```
static inline CCCryptorStatus ccInitCryptor
(CCCryptor *ref, const void *key, unsigned long key_len, const void *tweak_key, const void *iv

    size_t blocksize = ccGetCipherBlockSize(ref);
    uint8_t defaultIV[blocksize];


    if(iv == NULL) {
        CC_XZEROMEM(defaultIV, blocksize);
        iv = defaultIV;
    }

    ...

    return kCCSuccess;
}
```

While I am told this is probably common behavior in crypto libraries, I think it's dangerous. I ended up putting a comment in code examples warning developers about this behavior. So, heads up ;)

Source: parsiya.net/blog/2014-07-03-apples-common-crypto-library-defaults-to-a-zero-iv-if-one-is-not-provided/