Warmup: XOR it

- You are reverse-engineering a data format used for a mapping program, but the application seems to only output encrypted files!
- You see three things:
  - The output is called MAP.XML
  - The output is mostly binary nonsense, certainly not XML
  - You notice that the first 12 bytes of every file are the same
- How might you go about ‘decoding’ these files if you can’t reverse-engineer the program itself, but can ask it to generate new map files?

CSE 484: Computer Security and Privacy

Cryptography 3

Spring 2023

David Kohlbrenner

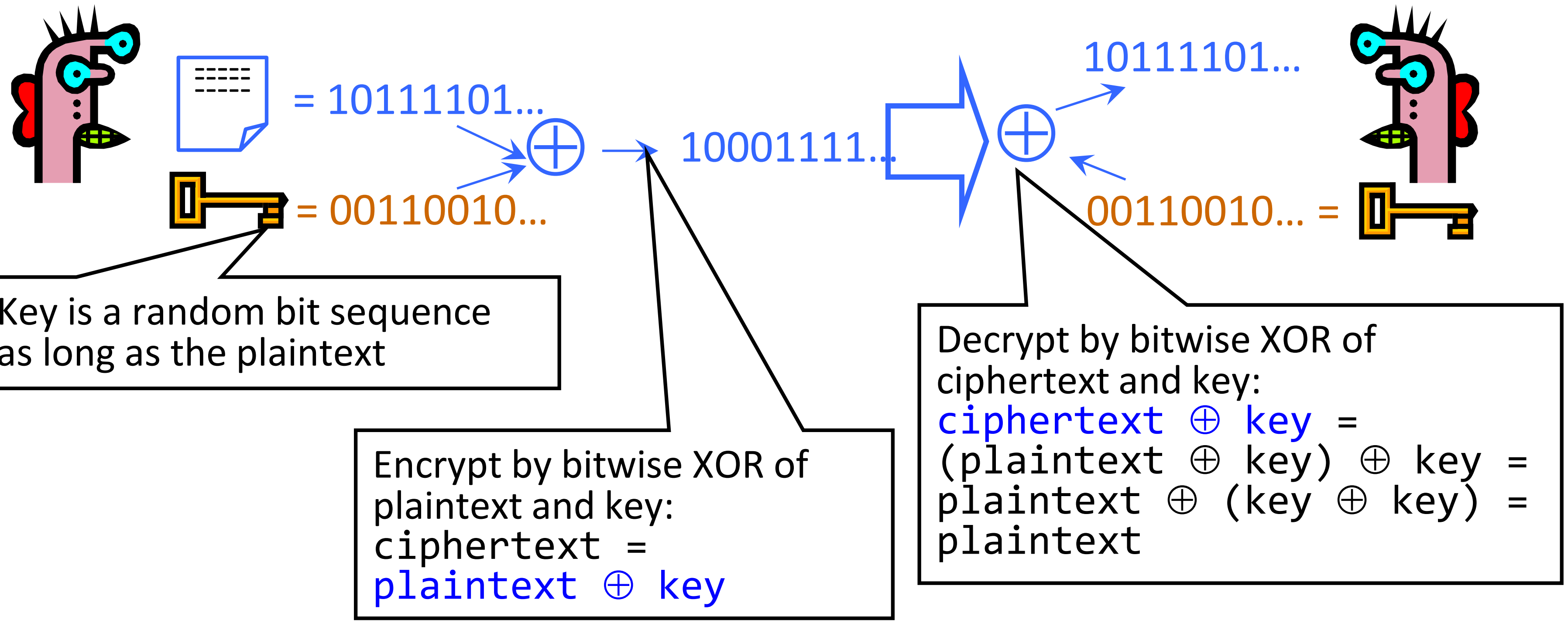
dkohlbre@cs

Thanks to Franzi Roesner, Dan Boneh, Dieter Gollmann, Dan Halperin, David Kohlbrenner, Yoshi Kohno, Ada Lerner, John Manferdelli, John Mitchell, Vitaly Shmatikov, Bennet Yee, and many others for sample slides and materials ...

Logistics

- Lab 1b coming up next week
- Homework 2 will go out by end of this week

One-Time Pad - Reminder



Cipher achieves perfect secrecy if and only if
there are as many possible keys as possible plaintexts,
and every key is equally likely (Claude Shannon, 1949)

Problems with One-Time Pad

- (1) Key must be as long as the plaintext
  - Impractical in most realistic scenarios
  - Still used for diplomatic and intelligence traffic
- (2) Insecure if keys are reused
  - Attacker can obtain XOR of plaintexts
- **(3) Does not guarantee integrity**
  - **One-time pad only guarantees confidentiality**
  - **Attacker cannot recover plaintext, but can easily change it to something else**

Reducing Key Size

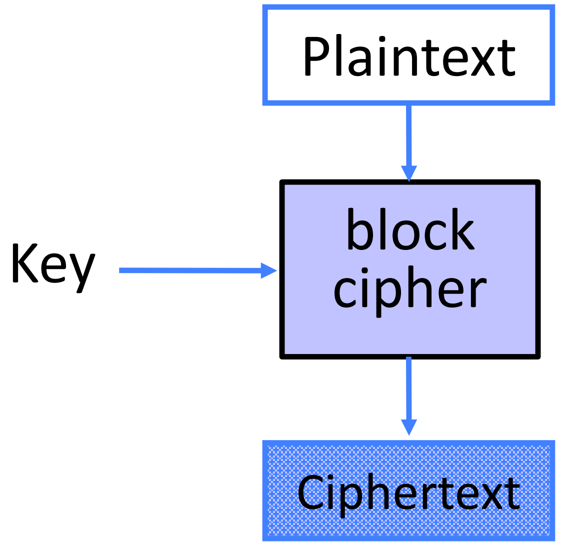
- What to do when it is infeasible to pre-share huge random keys?
  - When one-time pad is unrealistic…
- Use special cryptographic primitives: block ciphers, stream ciphers
  - Single key can be re-used (with some restrictions)
  - Not as theoretically secure as one-time pad

What if we try something simple?

- Alice and Bob synchronize their clocks perfectly, then generate OTPs
- Hash(time)
- Hash(time, key)

Block Ciphers

- Operates on a single chunk ("block") of plaintext
  - For example, 64 bits for DES, 128 bits for AES
  - Each key defines a different permutation
  - Same key is reused for each block (can use short keys)



Keyed Permutation

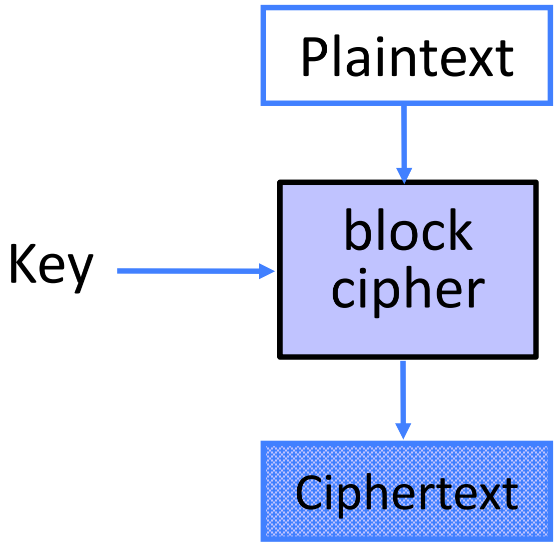
input	possible output	possible output	etc.
000	010	111	…
001	111	110	…
010	101	000	…
011	110	101	…
…	…		…
111	000	110	…

Key = 00
Key = 01

For N-bit input, 2^N! possible permutations
For K-bit key, 2^K possible keys

Keyed Permutation

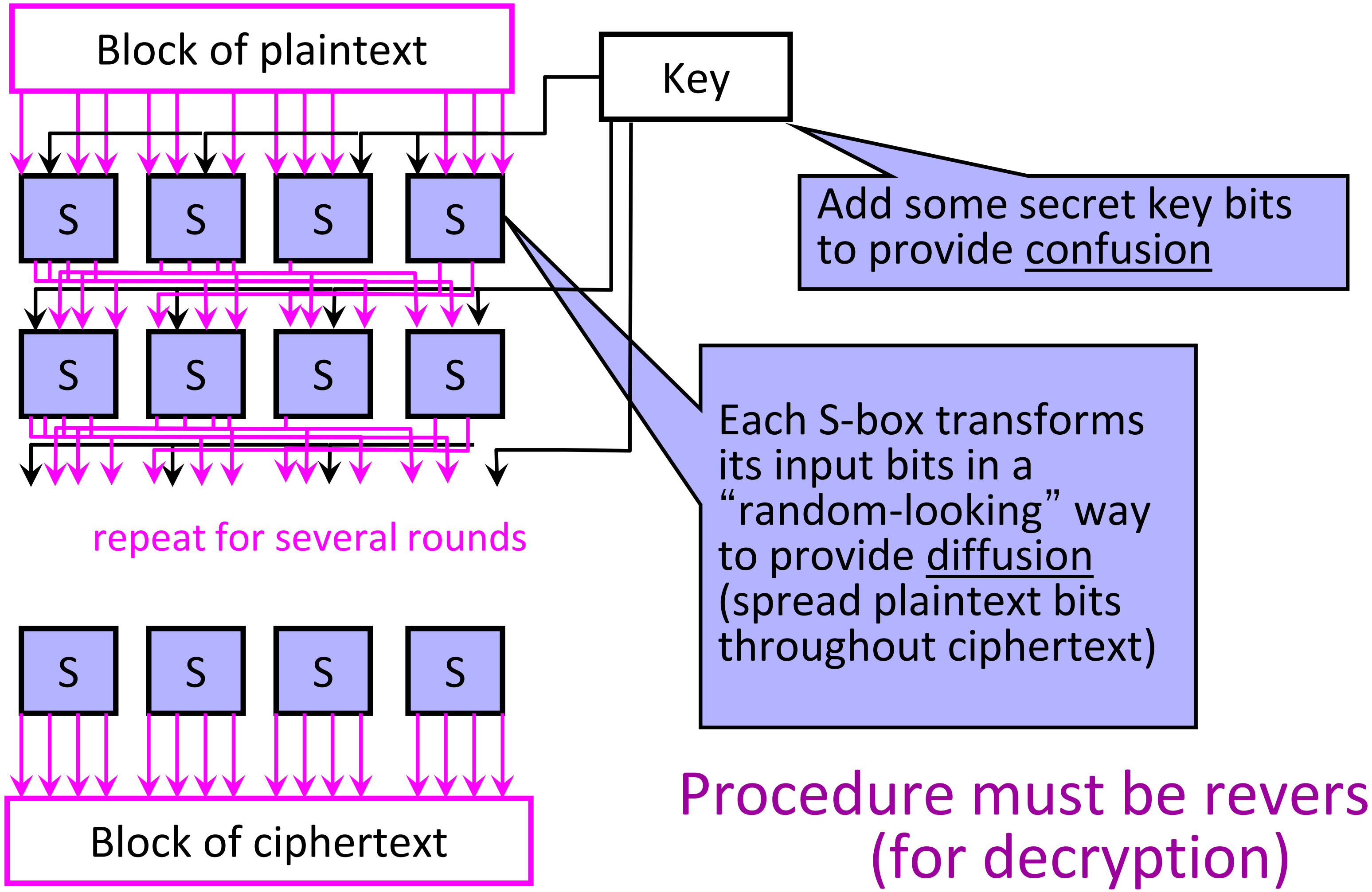
- Not just shuffling of input bits!
  - Suppose plaintext = “111”.
  - Then “111” is not the only possible ciphertext!
- Instead:
  - **Permutation of possible outputs**
  - Use secret key to pick a permutation



Block Cipher Security

- Result should look like a random permutation on the inputs
  - Recall: not just shuffling bits. N-bit block cipher permutes over 2^N inputs.
- Only computational guarantee of secrecy
  - Not impossible to break, just very expensive
    - If there is no efficient algorithm (unproven assumption!), then can only break by brute-force, try-every-possible-key search
  - Time and cost of breaking the cipher exceed the value and/or useful lifetime of protected information

Block Cipher Operation (Simplified)



Procedure must be reversible (for decryption)

Standard Block Ciphers

- **DES: Data Encryption Standard**
  - Feistel structure: builds invertible function using non-invertible ones
  - Invented by IBM, issued as federal standard in 1977
  - 64-bit blocks, 56-bit key + 8 bits for parity

DES and 56 bit keys

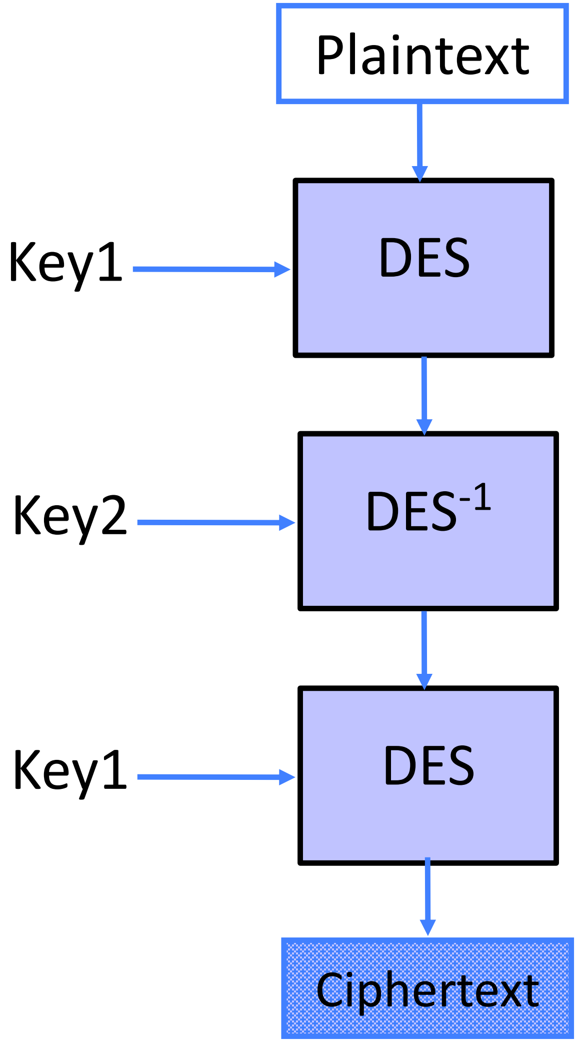
- 56 bit keys are quite short

Key Size (bits)	Number of Alternative Keys	Time required at 1 encryption/μs	Time required at 10^6 encryptions/μs
32	$2^{32} = 4.3 \times 10^9$	$2^{31}\ \mu s = 35.8$ minutes	2.15 milliseconds
56	$2^{56} = 7.2 \times 10^{16}$	$2^{55}\ \mu s = 1142$ years	10.01 hours
128	$2^{128} = 3.4 \times 10^{38}$	$2^{127}\ \mu s = 5.4 \times 10^{24}$ years	5.4×10^{18} years
168	$2^{168} = 3.7 \times 10^{50}$	$2^{167}\ \mu s = 5.9 \times 10^{36}$ years	5.9×10^{30} years
26 characters (permutation)	$26! = 4 \times 10^{26}$	$2 \times 10^{26}\ \mu s = 6.4 \times 10^{12}$ years	6.4×10^6 years

- 1999: EFF DES Crack + distributed machines
  - < 24 hours to find DES key
- DES ---> 3DES
  - 3DES: DES + inverse DES + DES (with 2 or 3 diff keys)

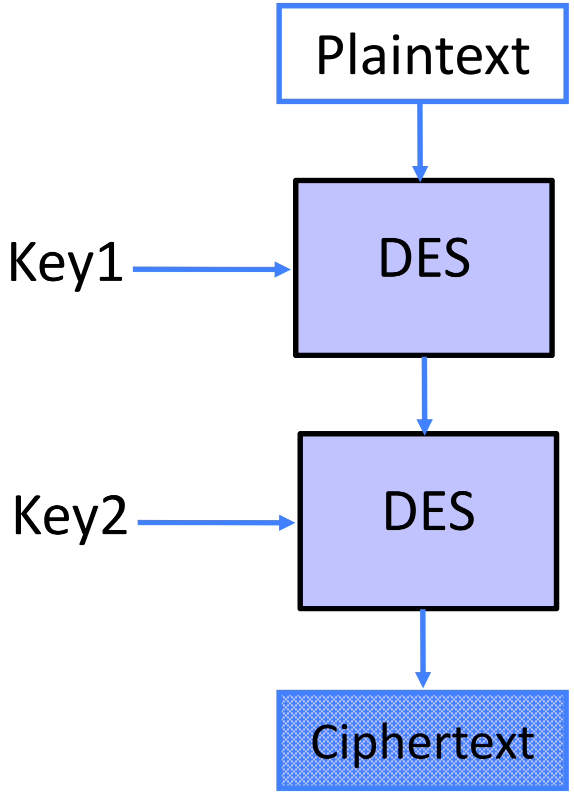
3DES

- Two-key 3DES increases security of DES by doubling the key length



But wait… what about *2DES*?

- Suppose you are given plaintext-ciphertext pairs (P1,C1), (P2,C2), (P3,C3)
- Suppose Key1 and Key2 are each 56-bits long
- Can you figure out Key1 and Key2 if you try all possible values for both (2^{112} possibilities) → Yes
- Can you figure out Key1 and Key2 more efficiently than that? → Discuss!

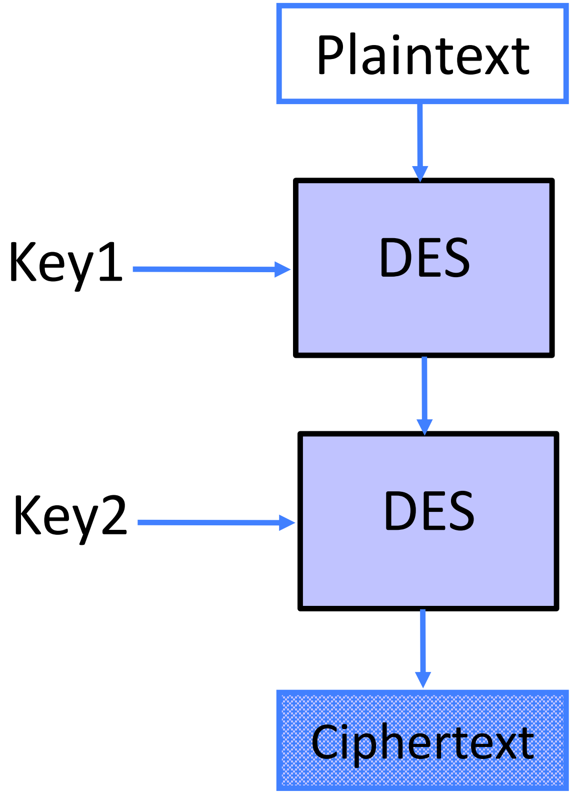


But wait… what about *2DES*?

- Meet-in-the-middle attack: guess K1 and K2 independently!

Meet-in-the-Middle Attack

- Guess 2^{56} values for Key1, and create a table from P1 to a middle value M1 for each key guess ($M1^{G1}$, $M1^{G2}$, $M1^{G3}$, …)
- Guess 2^{56} values for Key2, and create a table from C1 to a middle value M'1 for each key guess ($M'1^{G1}$, $M'1^{G2}$, $M'1^{G3}$, …)
- Look for collision in the middle values → if only one collision, found Key1 and Key2; otherwise repeat for (P2,C2), …



Defining the strength of a scheme

- *Effective Key Strength*
  - Amount of 'work' the adversary needs to do
- DES: 56-bits
  - 2^56 encryptions to try 'all keys'
- 2DES: 57-bits
  - 2*(2^56) encryptions = 2^57
- 3DES: 112-bits (or sometimes 80-bits)
  - Meet-in-the-middle + more work = 2^112 (for 3 keys, e.g. K1, K2, K3)
  - Various attacks = 2^80 (for 2 keys, e.g. K1, K2, K1)

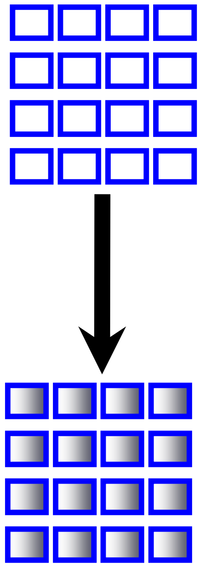
Standard Block Ciphers

- **DES: Data Encryption Standard**
  - Feistel structure: builds invertible function using non-invertible ones
  - Invented by IBM, issued as federal standard in 1977
  - 64-bit blocks, 56-bit key + 8 bits for parity

- **AES: Advanced Encryption Standard**
  - New federal standard as of 2001
    - NIST: National Institute of Standards & Technology
  - Based on the Rijndael algorithm
    - Selected via an open process
  - 128-bit blocks, keys can be 128, 192 or 256 bits

Encrypting a Large Message

- So, we've got a good block cipher, but our plaintext is larger than 128-bit block size



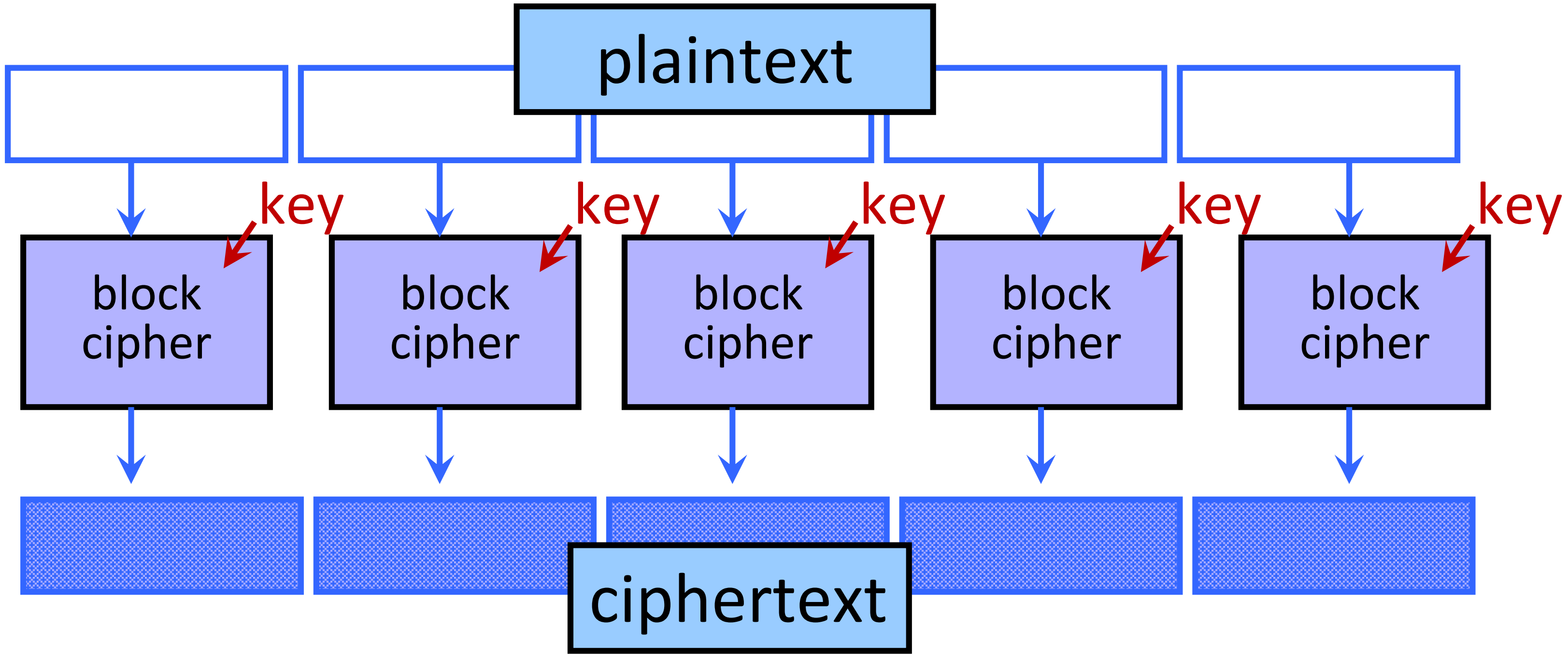
128-bit plaintext

(arranged as 4x4 array of 8-bit bytes)

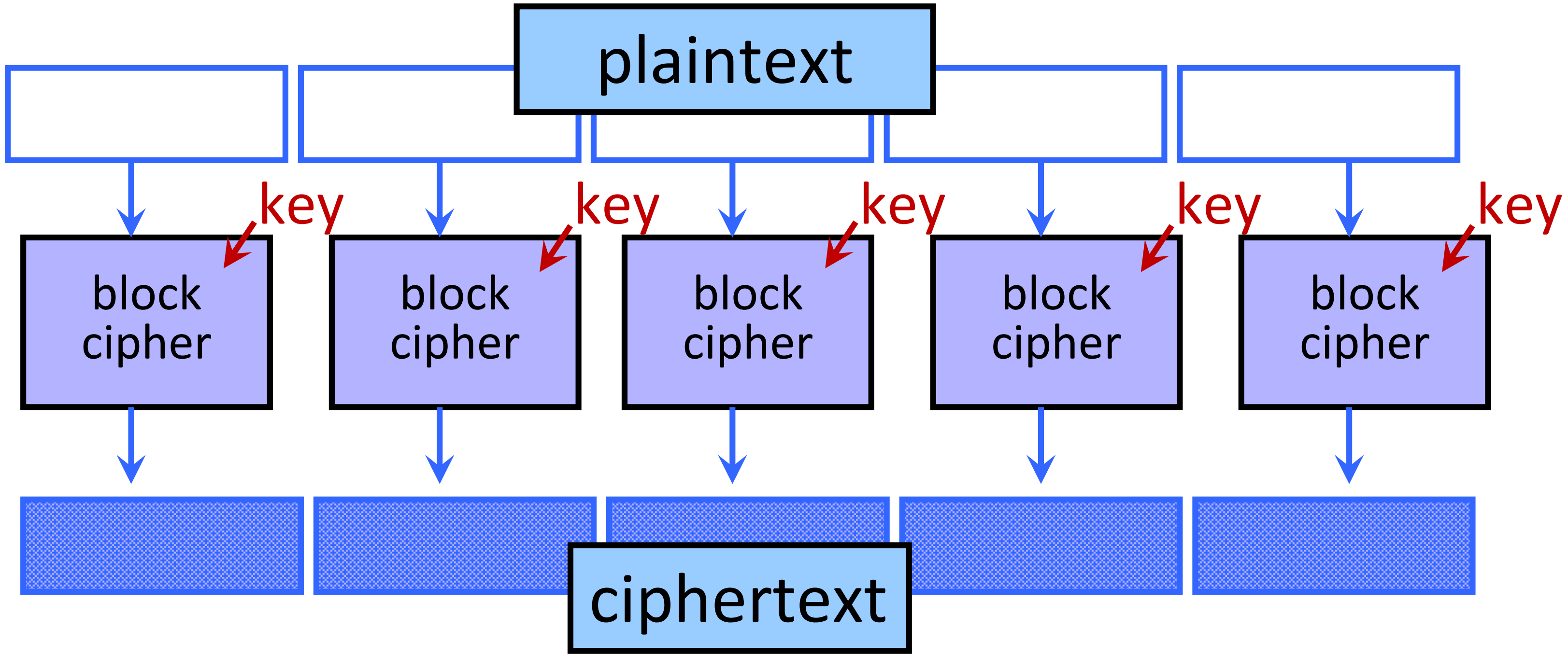
128-bit ciphertext

- What should we do?

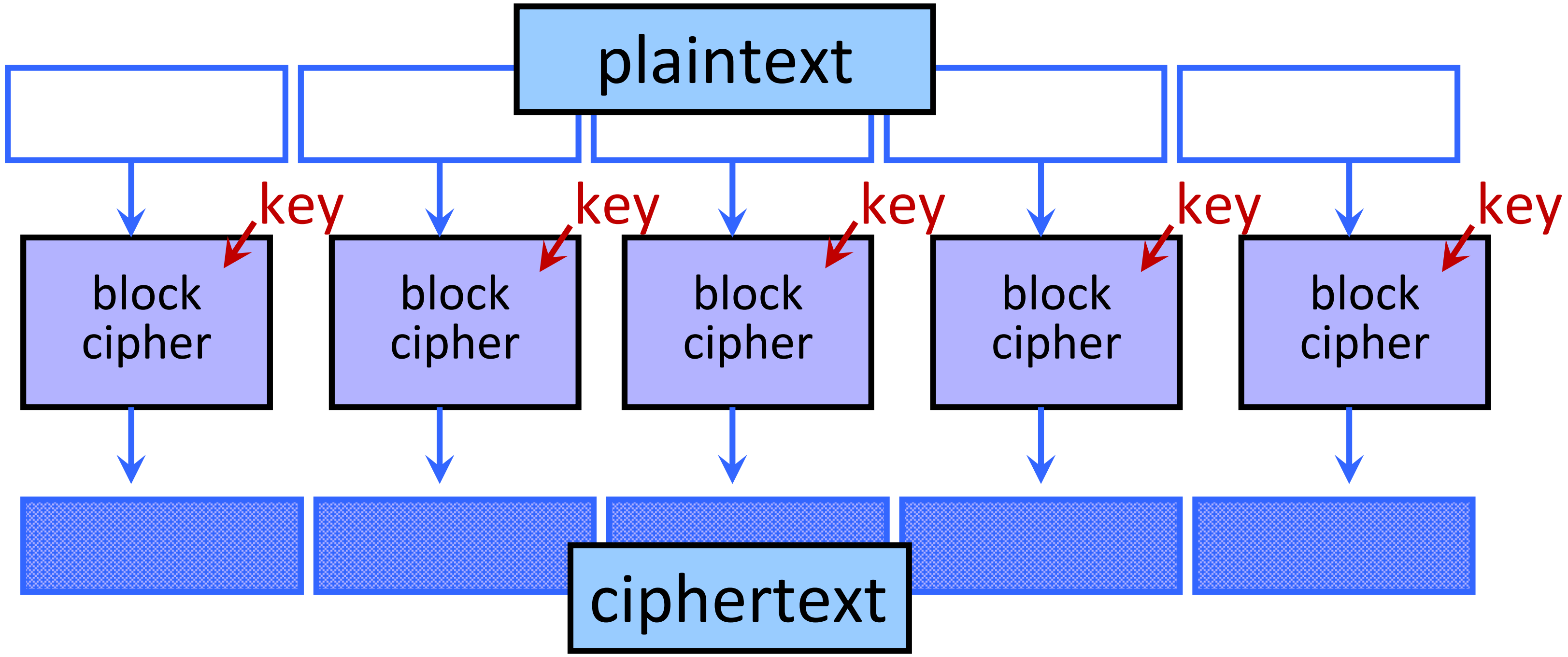
Electronic Code Book (ECB) Mode



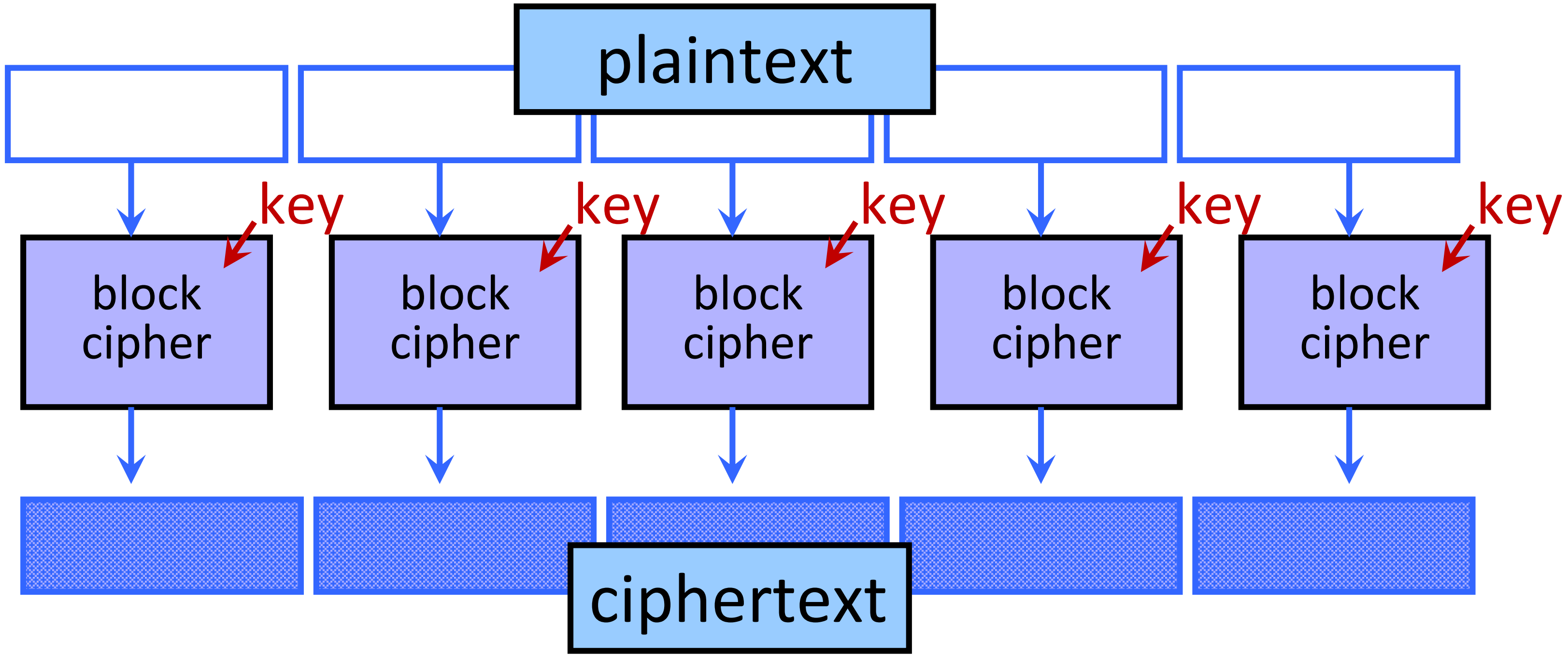
Electronic Code Book (ECB) Mode



Canvas: What properties of ECB aren't great?

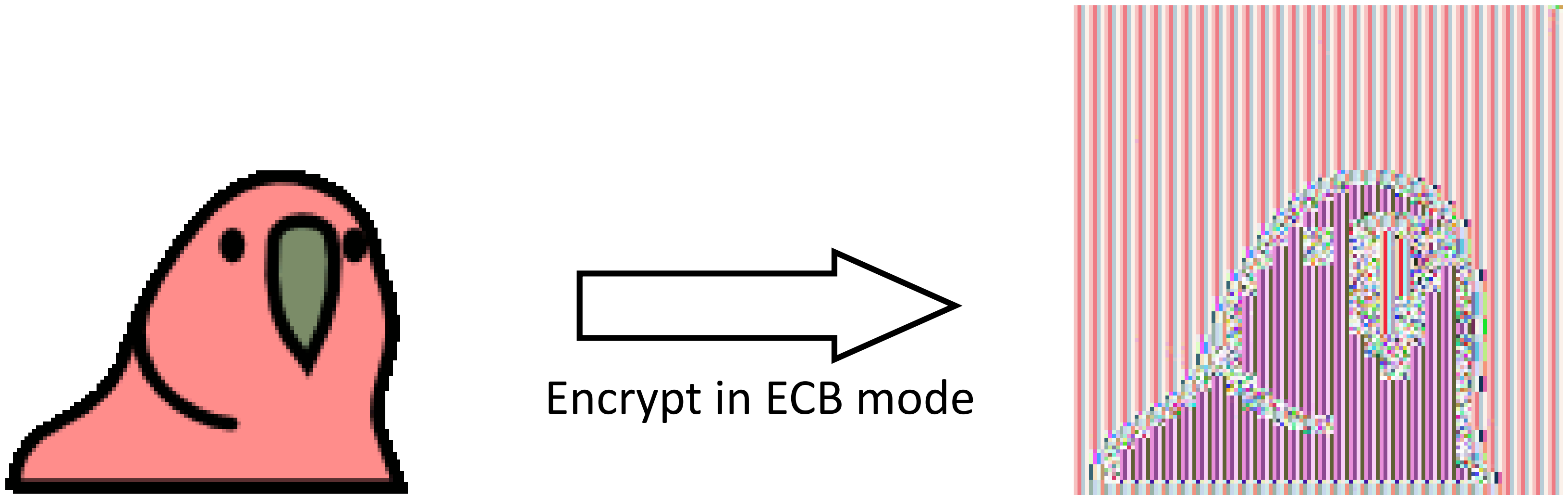


Electronic Code Book (ECB) Mode



- Identical blocks of plaintext produce identical blocks of ciphertext
- No integrity checks: can mix and match blocks

Information Leakage in ECB Mode



Oops

Move Fast and Roll Your Own Crypto

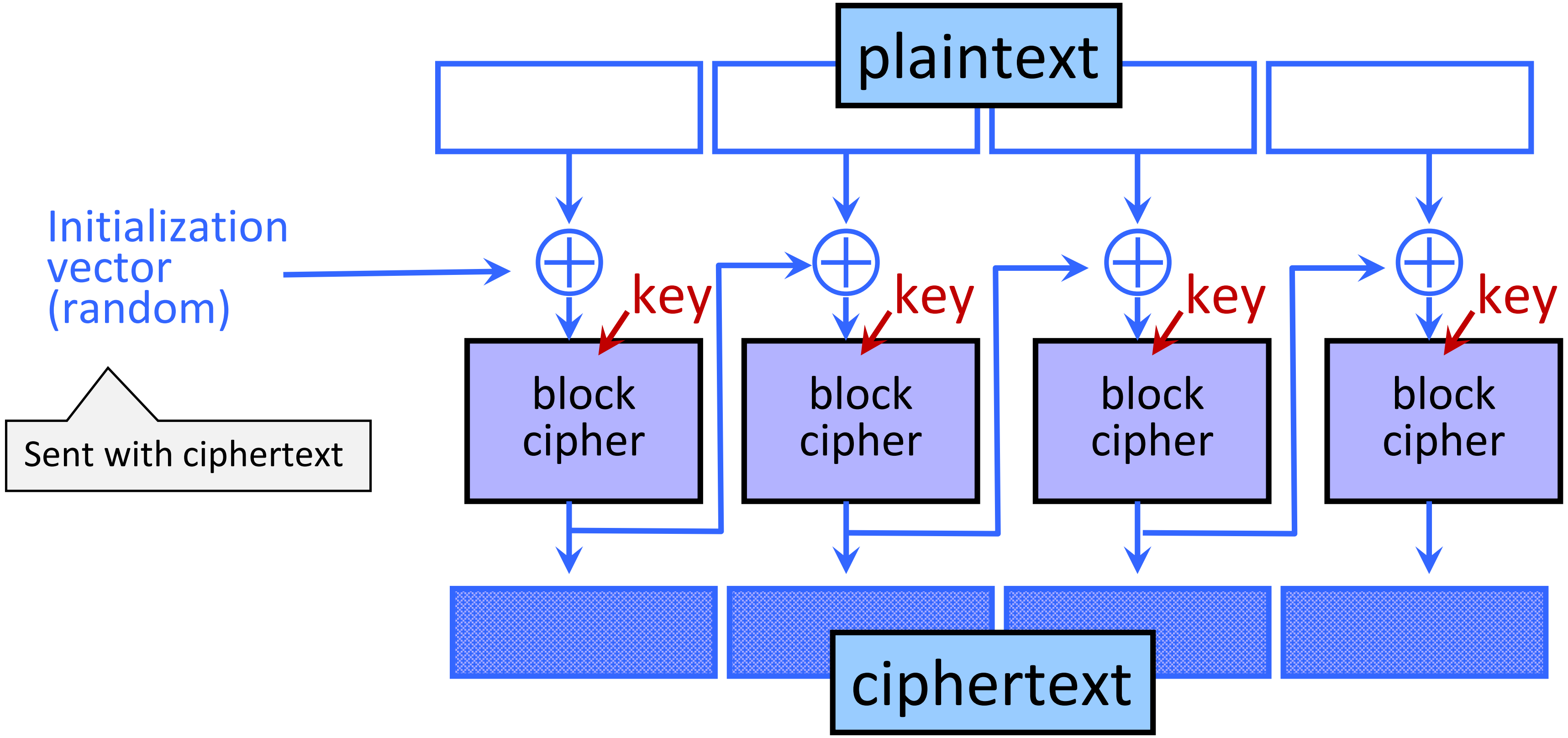
A Quick Look at the Confidentiality of Zoom Meetings

By Bill Marczak and John Scott-Railton April 3, 2020

- Zoom documentation claims that the app uses “AES-256” encryption for meetings where possible. However, we find that in each Zoom meeting, a single AES-128 key is used in ECB mode by all participants to encrypt and decrypt audio and video. The use of ECB mode is not recommended because patterns present in the plaintext are preserved during encryption.

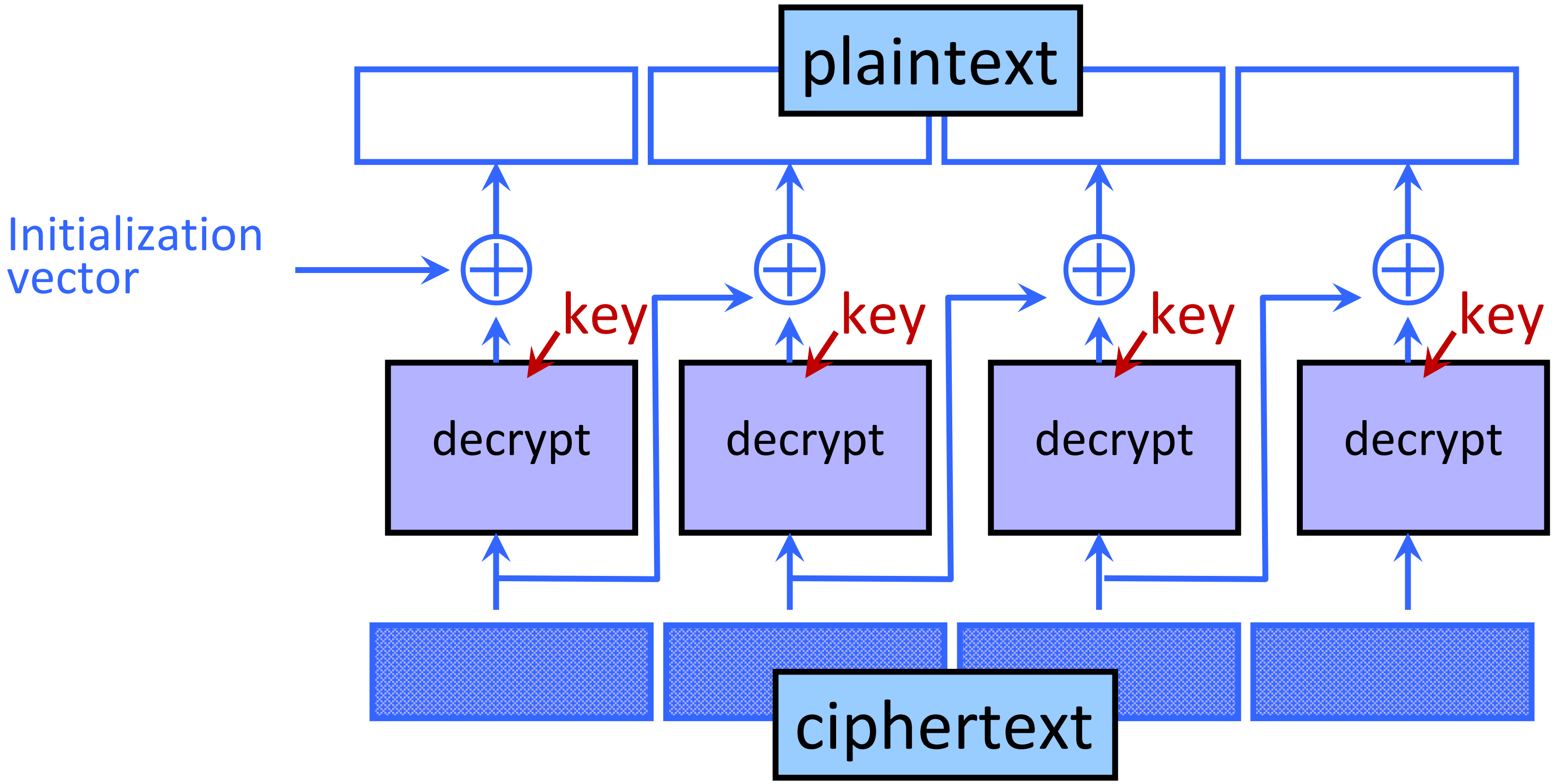
https://citizenlab.ca/2020/04/move-fast-roll-your-own-crypto-a-quick-look-at-the-confidentiality-of-zoom-meetings/

Cipher Block Chaining (CBC) Mode: Encryption

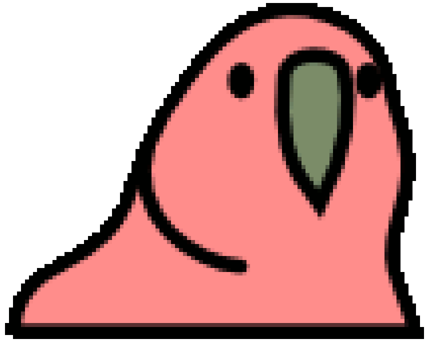


- Identical blocks of plaintext encrypted differently
- Last cipherblock depends on entire plaintext
  - Still does not guarantee integrity

CBC Mode: Decryption

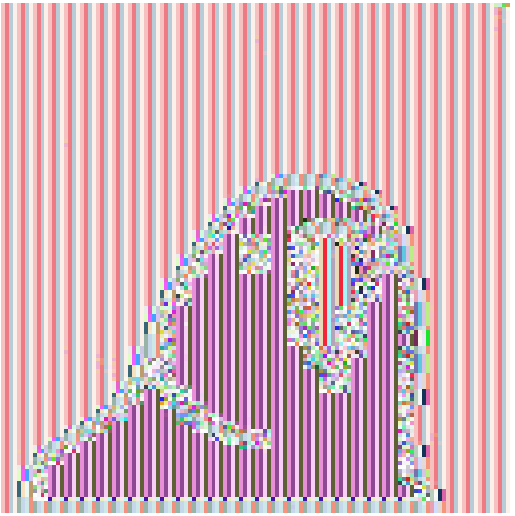


ECB vs. CBC

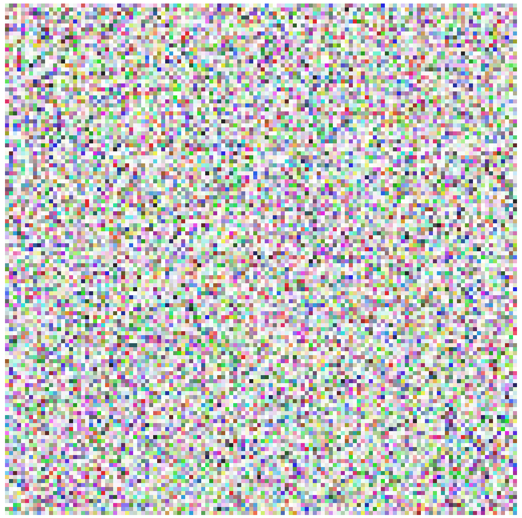


AES in ECB mode

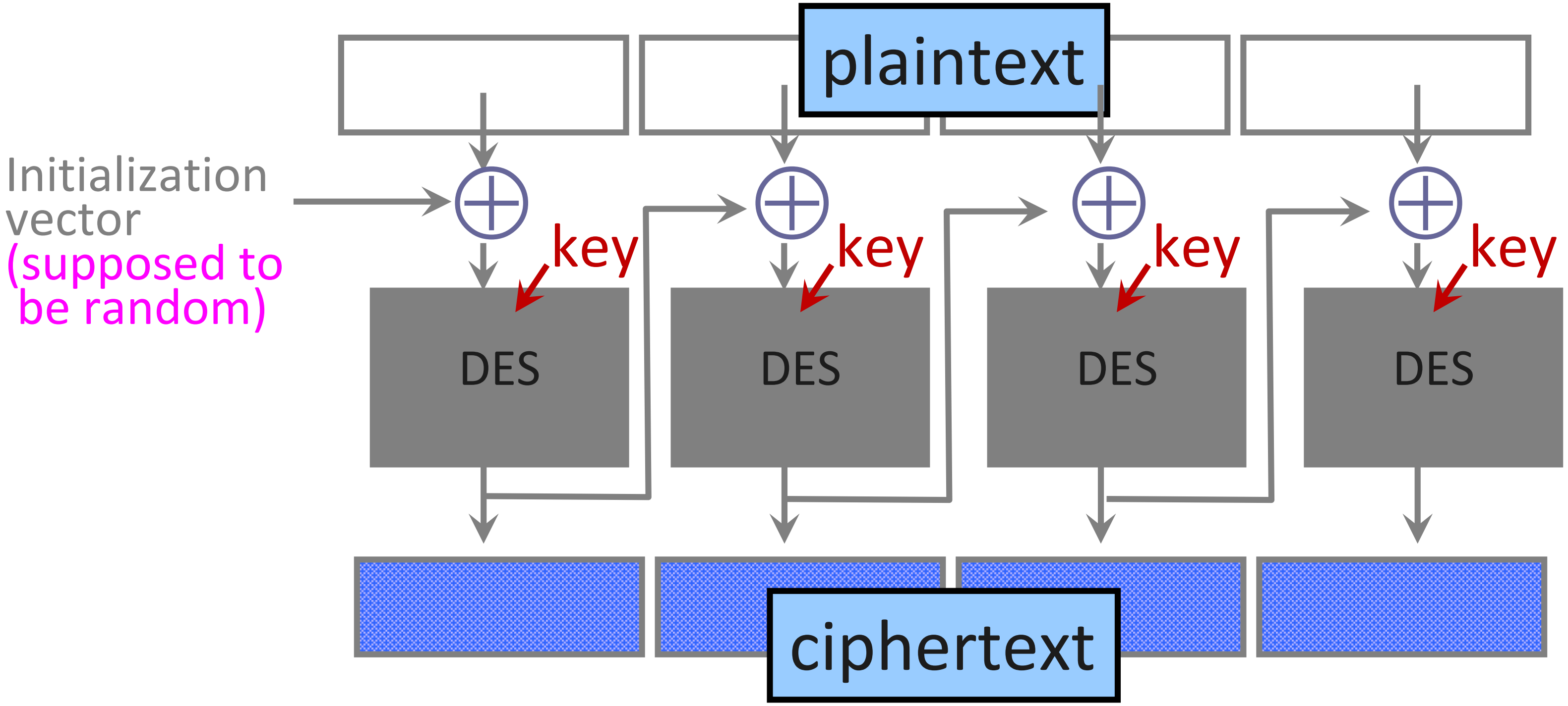
AES in CBC mode



Similar plaintext
blocks produce
similar ciphertext
blocks (not good!)



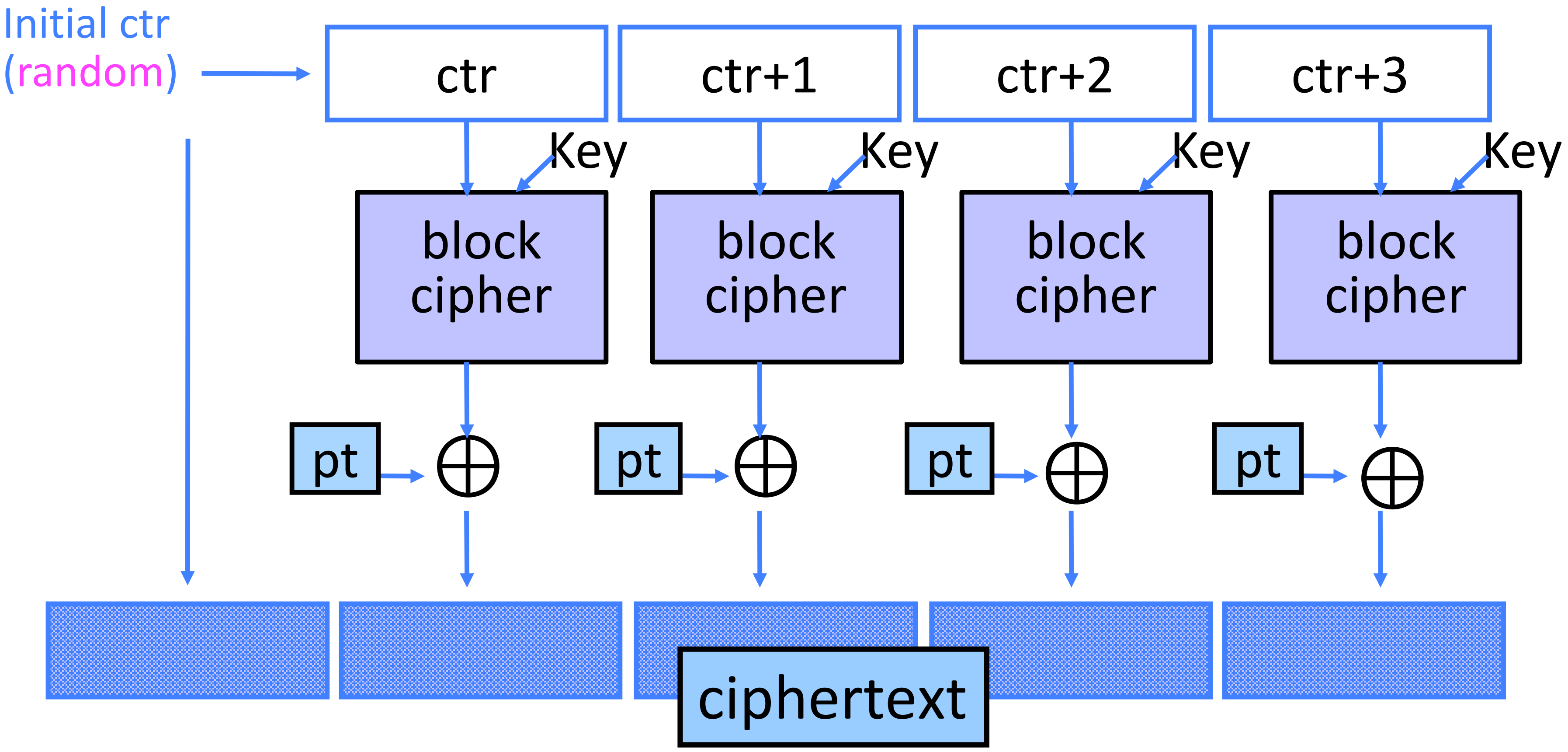
Initialization Vector Dangers



Found in the source code for Diebold voting machines:

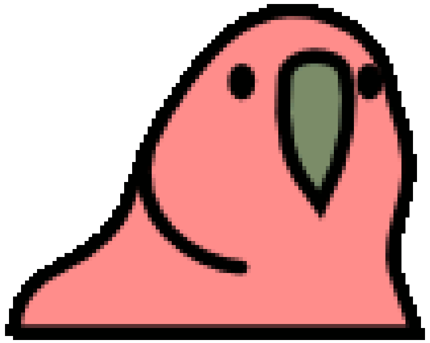
```
DesCBCEncrypt((des_c_block*)tmp, (des_c_block*)record.m_Data,
        totalSize, DESKEY, NULL, DES_ENCRYPT)
```

Counter Mode (CTR): Encryption



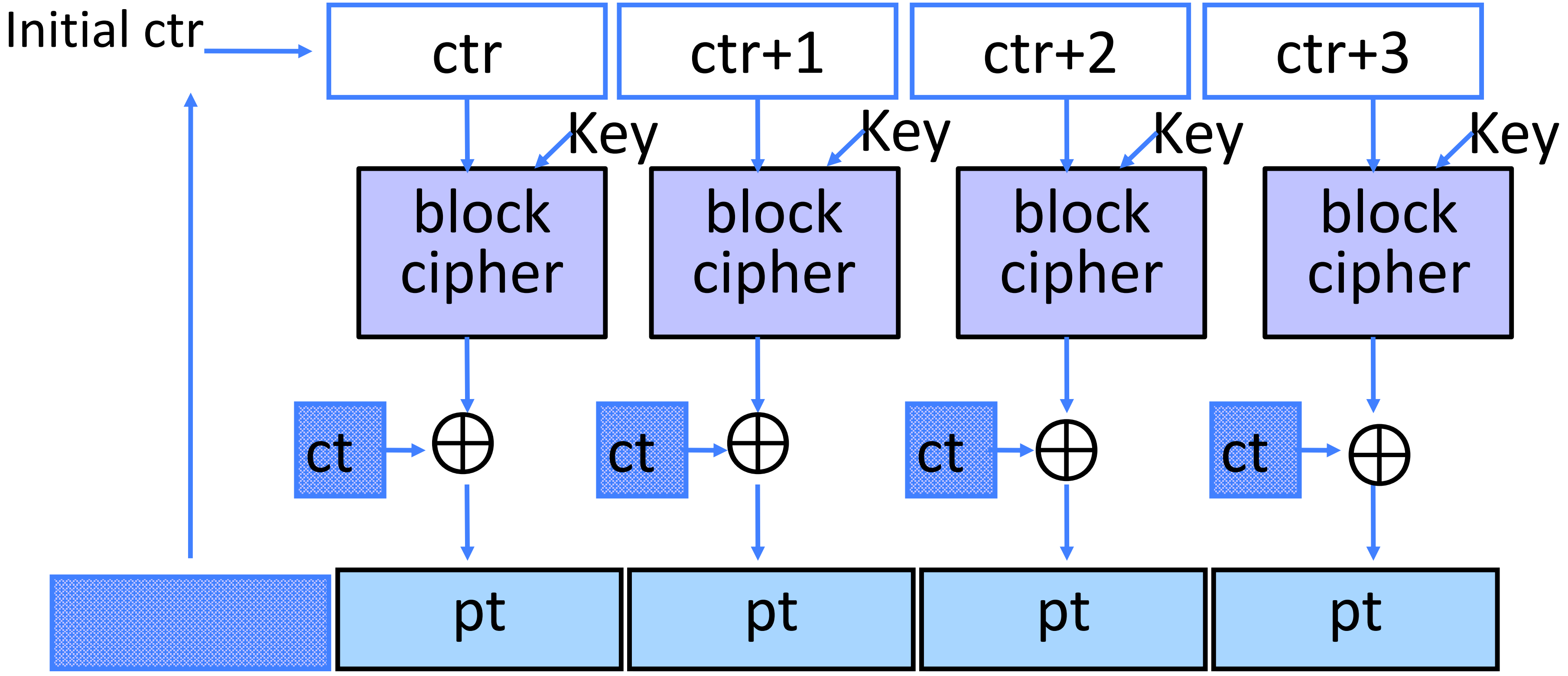
- Identical blocks of plaintext encrypted differently
- Still does not guarantee integrity; Fragile if ctr repeats

Information Leakage in CTR Mode (poorly)



Encrypt in CTR mode:
But with the same counter for each frame!

Counter Mode (CTR): Decryption



Ok, so what mode do I use?

- Don’t choose a mode, use established libraries ☺
- Good modes:
  - GCM - Galois/Counter Mode
  - CTR (sometimes)
  - Even ECB is fine in ‘the right circumstance’
- AES-128 is standard
  - Be concerned if something says “AES 1024”…

https://research.kudelskisecurity.com/2022/05/11/practical-bruteforce-of-aes-1024-military-grade-encryption/

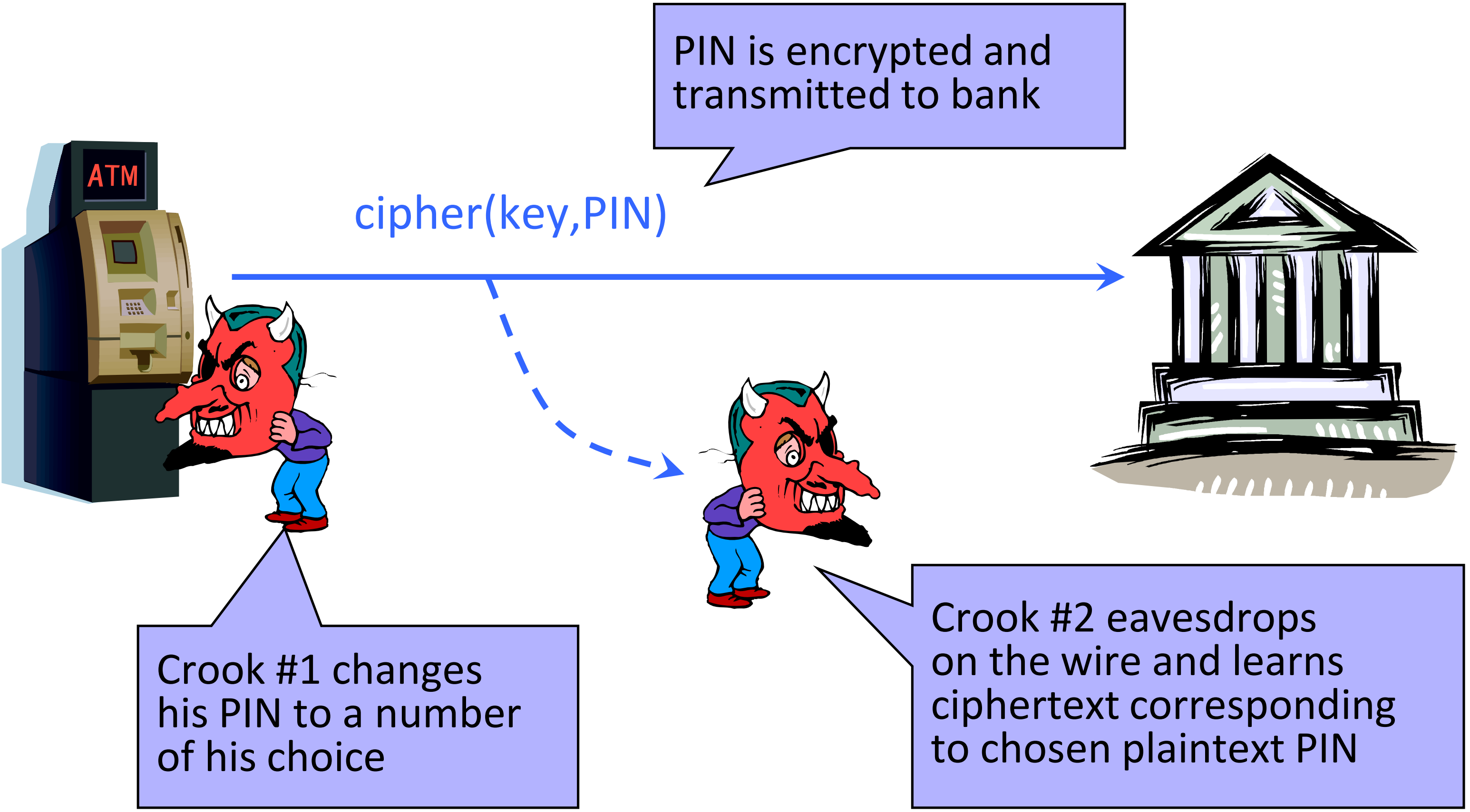
When is an Encryption Scheme "Secure"?

- Hard to recover the key?
  - What if attacker can learn plaintext without learning the key?
- Hard to recover plaintext from ciphertext?
  - What if attacker learns some bits or some function of bits?

How Can a Cipher Be Attacked?

- Attackers knows ciphertext and encryption algorithm
  - What else does the attacker know? Depends on the application in which the cipher is used!
- Ciphertext-only attack
- KPA: Known-plaintext attack (stronger)
  - Knows some plaintext-ciphertext pairs
- CPA: Chosen-plaintext attack (even stronger)
  - Can obtain ciphertext for any plaintext of choice
- CCA: Chosen-ciphertext attack (very strong)
  - Can decrypt any ciphertext <u>except</u> the target

Chosen Plaintext Attack



… repeat for any PIN value

Very Informal Intuition

Minimum security requirement for a modern encryption scheme

- Security against chosen-plaintext attack (CPA)
  - Ciphertext leaks no information about the plaintext
  - Even if the attacker correctly guesses the plaintext, he cannot verify his guess
  - Every ciphertext is unique, encrypting same message twice produces completely different ciphertexts
    - Implication: encryption must be randomized or stateful
- Security against chosen-ciphertext attack (CCA)
  - Integrity protection – it is not possible to change the plaintext by modifying the ciphertext

The shape of the formal approach

- INDistinguishability under Chosen Plaintext Attack
  - IND-CPA
- Formalized *cryptographic game*

- Adversary submits pairs of *plaintexts* (M_a, M_b)
  - Gets back ONE of the *ciphertexts* (C_x)

- Adversary must guess which ciphertext this is (C_a or C_b)
  - If they can do better than 50/50, they win