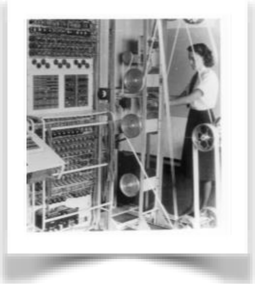
Fundamentals of Computer Security

Fall 2022

Radu Sion

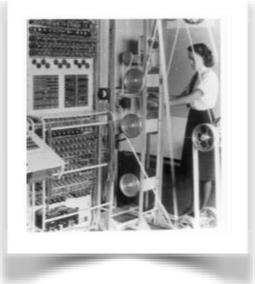
Symmetric-key Encryption
Ciphers

Thanks to Ari Juels for parts of this deck!



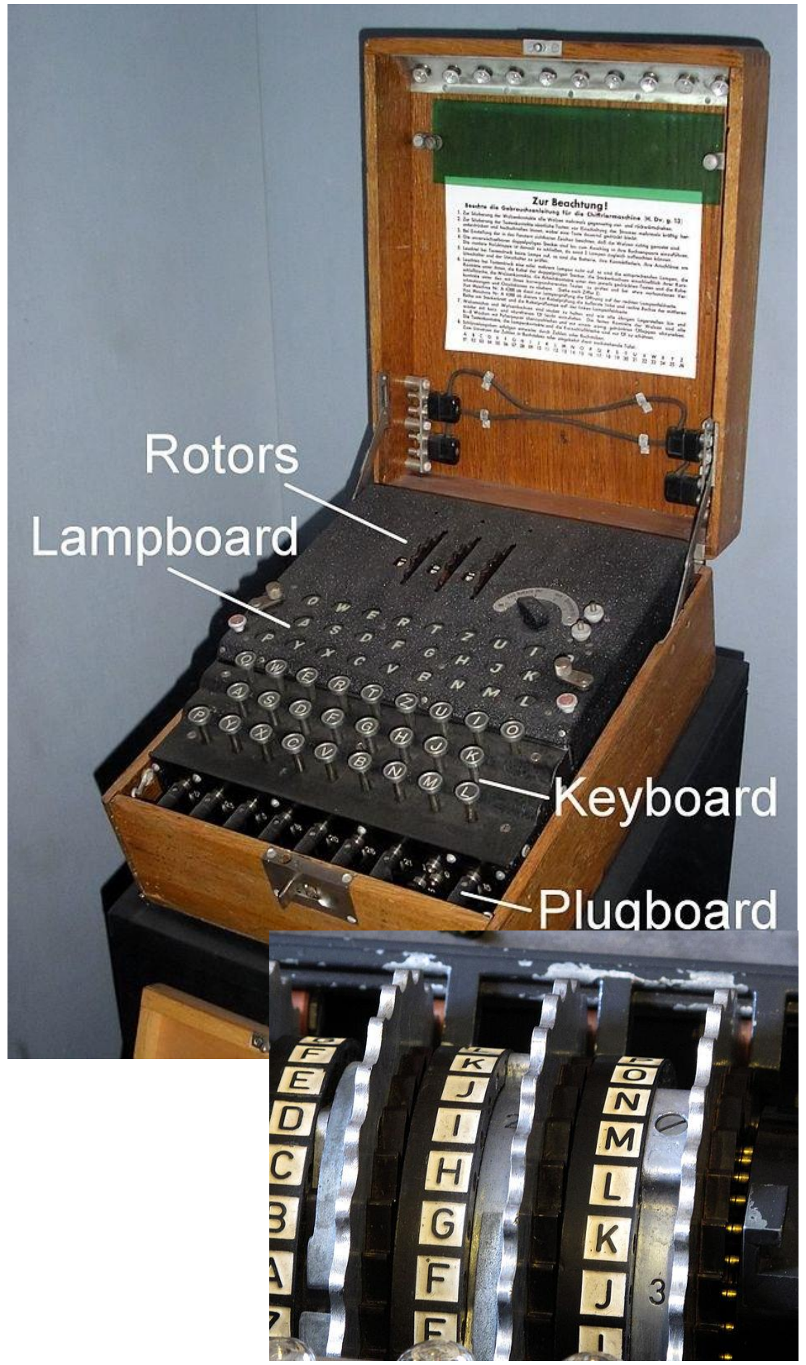
The modern computer

- In early history, people communicated at a distance via letters, messengers.. eventually telegraph
- Radio communication grew in the early 20th century; very convenient, but…
- Everyone could hear and eavesdrop on your transmissions!
  - Radio changed the **adversarial model**!
- Especially during wartime, encryption became important.
- WWI hand ciphers gave way in WWII to cipher machines…

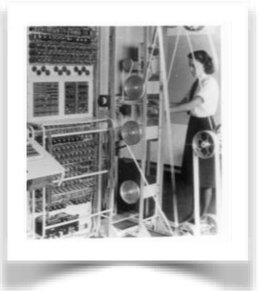


Enciphering machines

- During WWII, the Germans used machines in the Enigma family.
- These machines enciphered using electromechanical rotors.
- The Enigmas had many possible settings…
- An Allied cryptanalyst faced in practice an estimated 10^{23} possible settings.
  - That's a hundred thousand billion billion!



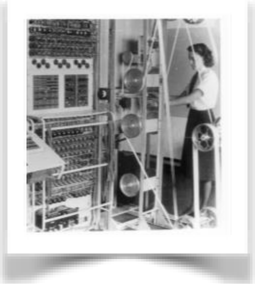
German Enigma machine



How were these broken?

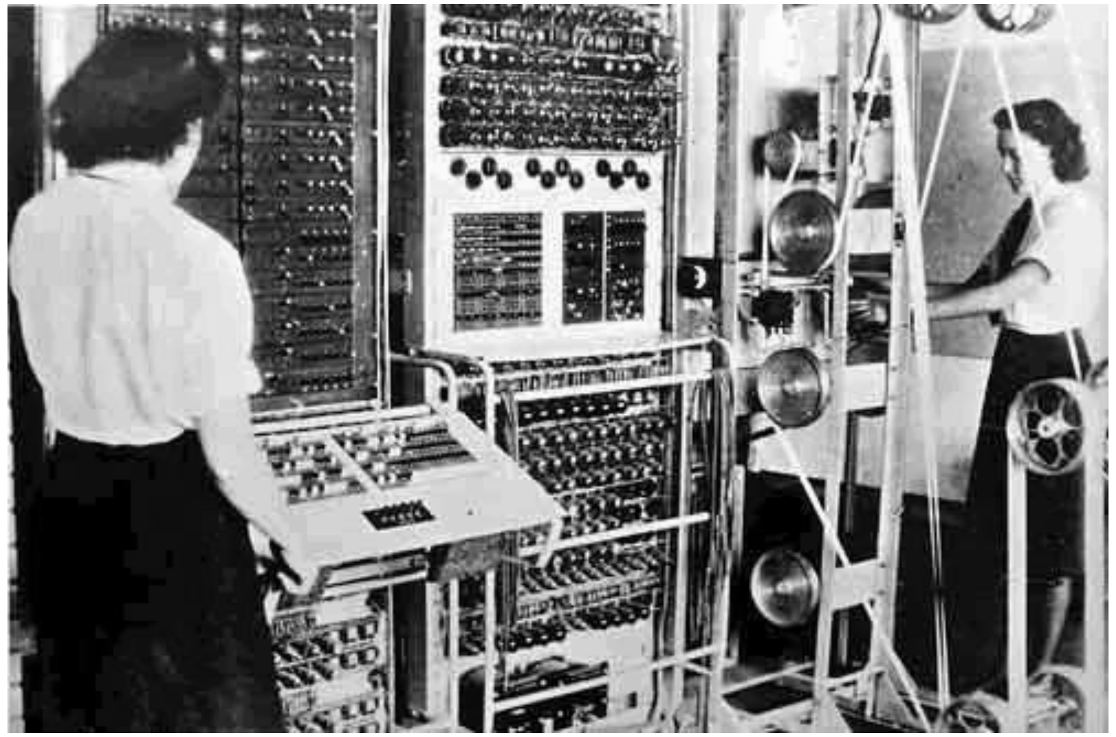
- "Bombes" were developed by British cryptologists to simulate Engima behavior.
  - Initial design by **Alan Turing**
  - A kind of proto-computer
- Bombes explored Enigma daily settings (the set and positions of rotors, the key, and the plugboard wirings).
- They enabled effective breaks of Enigma-encoded messages: yielded part of the ULTRA intelligence that played an enormous part in Allied victories.
- Seen *The Imitation Game*?

Bombe reconstruction at Bletchley Park



Colossus

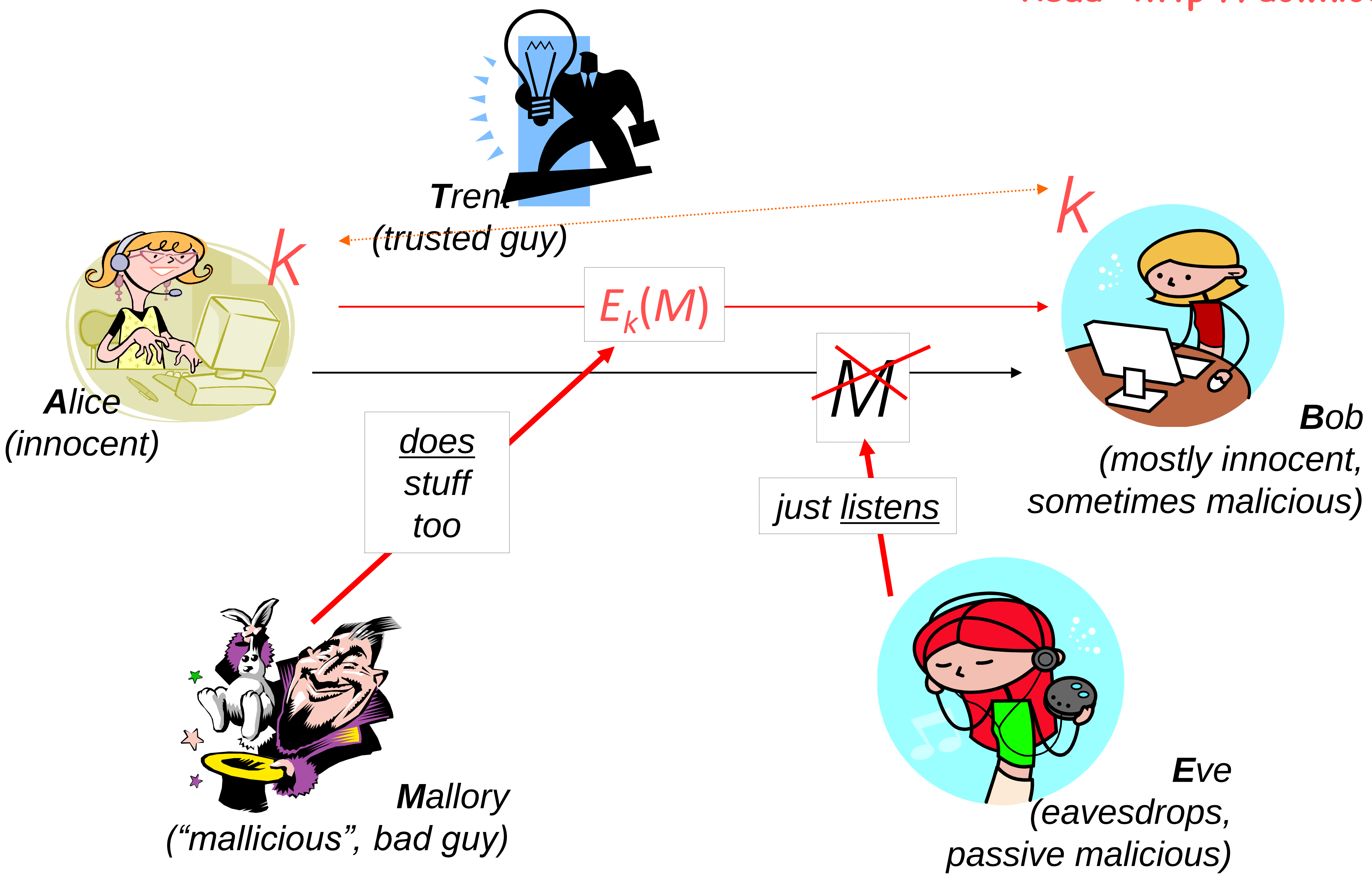
- Another component of ULTRA was the Colossus machine.
  - Used to attack the Lorenz SZ40/42 in-line cipher machine, not Enigma.
- It was the world's first programmable electronic digital computing machine.
- **Codebreaking—infosec again—was intimately bound up in the birth of the programmable digital computer.**



A Colossus Mark 2 computer being operated by Dorothy Du Boisson and Elsie Booker (1944-5) [U.K. National Archives, FO850/234]

Meet the Cast

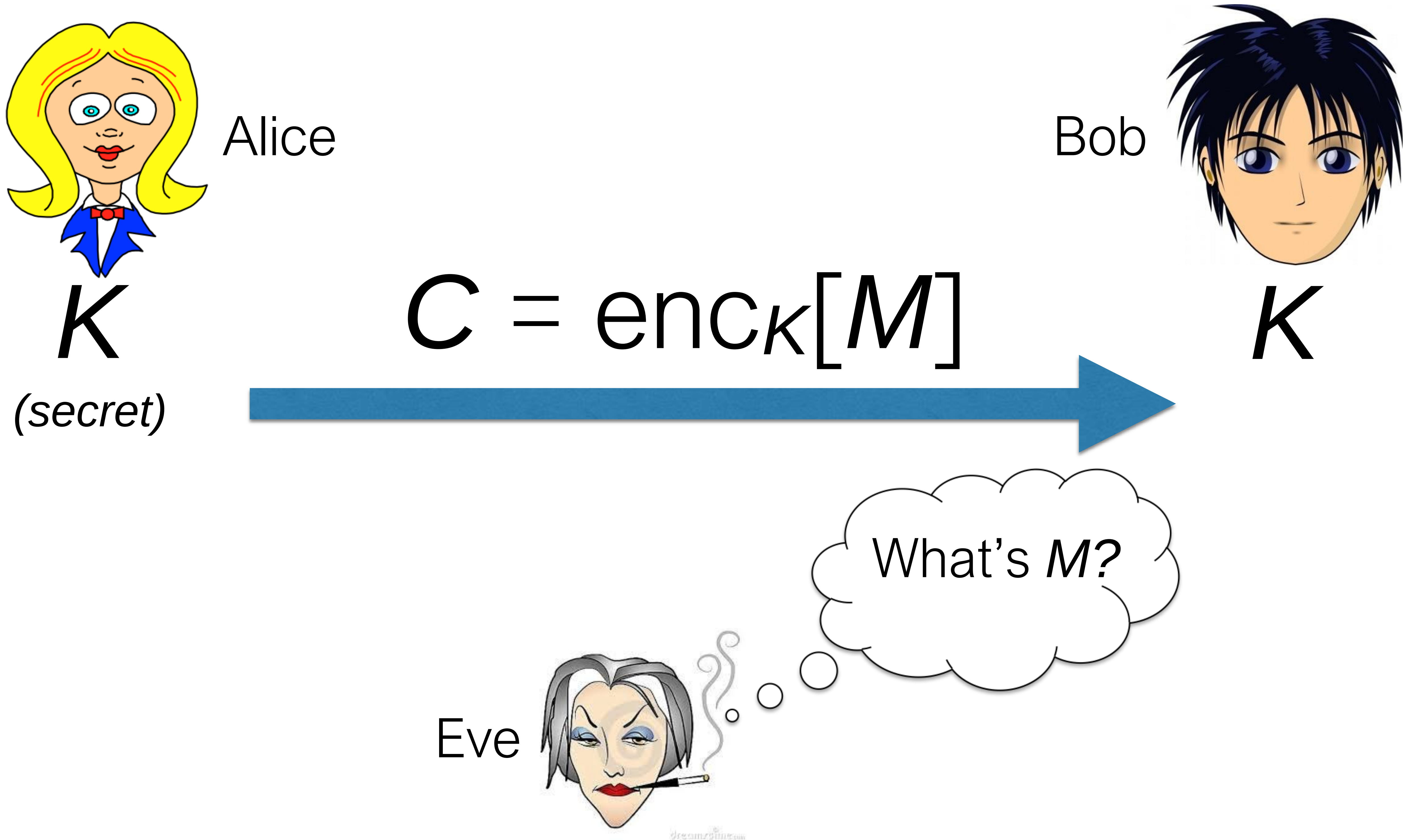
Read: http://downlode.org/etext/alicebob.html !



An inconvenient truth

- Where does *k* come from ? (“key distribution”)
- Can Eve distinguish between $E_k(M_1)$ and $E_k(M_2)$ if she knows M_1 and M_2 ? Should not be able to !!! (“semantic security”)
- Make sure that $E_k(M_1) \neq E_k(M_2)$ if $M_1 \neq M_2$ (maybe not ?)
- Can Mallory modify $E_k(M)$ into an $E_k(M_{mallory})$? (”malleability”)
- etc (! lots of stuff !)
- Danger: things seem trivial and they are not – result: super weak systems !

Symmetric-key encryption



Caesar Cipher

- Example: Cæsar cipher
  - M = { sequences of letters }
  - K = { i | i is an integer and $0 \leq i \leq 25$ }
  - E = { E_k | $k \in$ K and for all letters m,

    $E_k(m) = (m + k) \bmod 26$ }
  - D = { D_k | $k \in$ K and for all letters c,

    $D_k(c) = (26 + c - k) \bmod 26$ }
  - C = M

Attacks

- Opponent whose goal is to break cryptosystem is the *adversary*
  - Assume adversary knows algorithm used, but not key
- Many types of attacks:
  - *ciphertext only*: adversary has only ciphertext; goal is to find plaintext, possibly key
  - *known plaintext*: adversary has ciphertext, corresponding plaintext; goal is to find key
  - *chosen plaintext*: adversary may supply plaintext and obtain corresponding ciphertext; goal is to find key
  - *chosen ciphertext*: adversary may supply ciphertext and obtain corresponding plaintext; goal is to find key
  - etc

How to attack?

- Mathematical attacks
  - Based on analysis of underlying mathematics
- Statistical attacks
  - Make assumptions about the distribution of letters, pairs of letters (digrams), triplets of letters (trigrams), *etc.*
    - Called *models of the language*
  - Examine ciphertext, correlate properties with the assumptions.

Statistics

- Compute frequency of each letter in ciphertext:

G	0.1	H	0.1	K	0.1	O	0.3
R	0.2	U	0.1	Z	0.1		

- Apply 1-gram model of English
- Correlate and invert encryption

Caesar has a Problem ☺

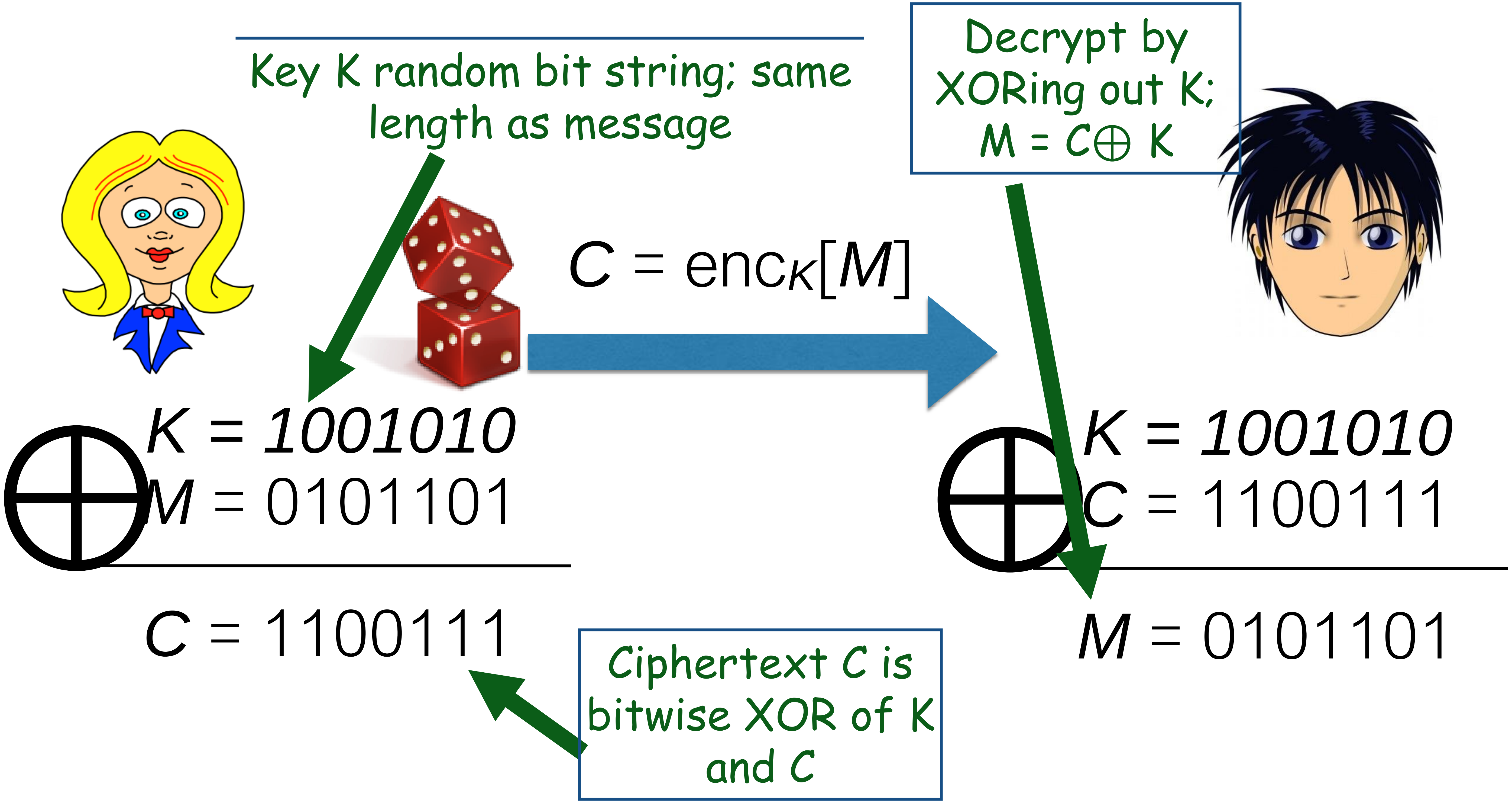
- Key is too short
  - Can be found by exhaustive search
  - Statistical frequencies not concealed well
    - They look too much like regular English letters
- So make it longer
  - Multiple letters in key
  - Idea is to smooth the statistical frequencies to make cryptanalysis harder

Vigènere Cipher

- Like Cæsar cipher, but use a phrase
- Documented by Blaise de Vigenere (court of Henry III of France) in Paris, 1586 – actually a variant of a cipher by a J.B. Porter
- Example
  - Message THE BOY HAS THE BALL
  - Key VIG
  - Encipher using Cæsar cipher for each letter:

key VIGVIGVIGVIGVIGV
plain THEBOYHASTHEBALL
cipher OPKWWECIYOPKWIRG

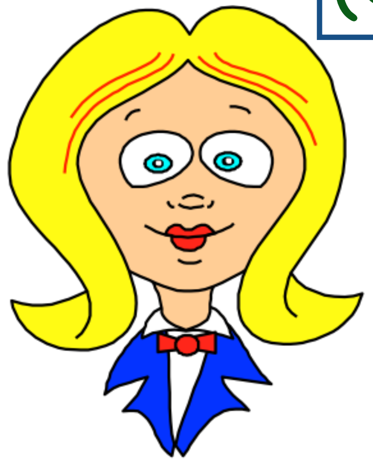
"Unbreakable" cipher: One-time pad



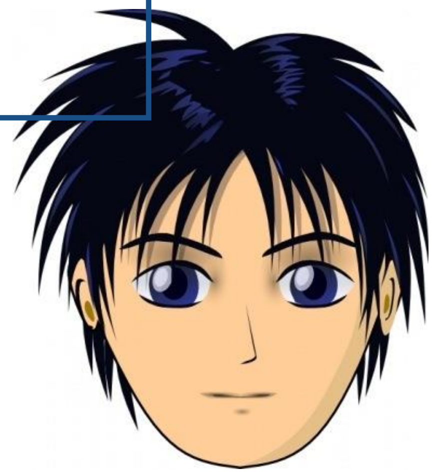
One-time pad

Perfect secrecy if every K equally likely… because:
* For any M, every possible C equally likely!
* So C reveals no information about M!

(C. Shannon, 1949)



$C = \text{enc}_K[M]$

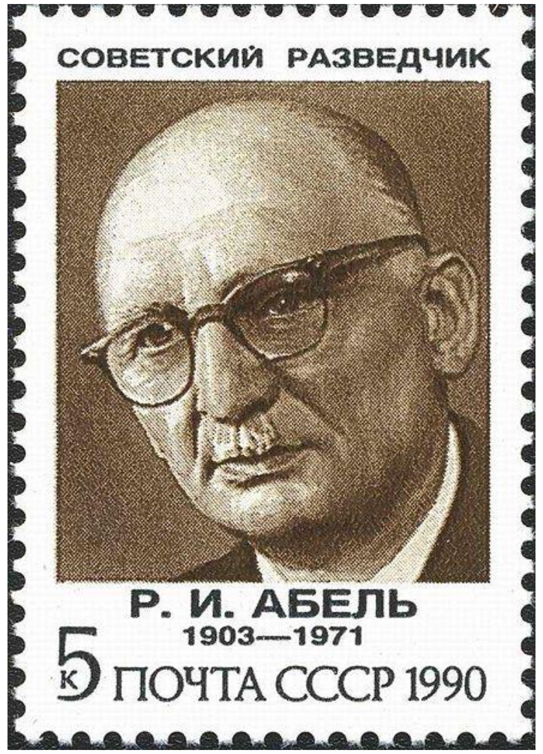


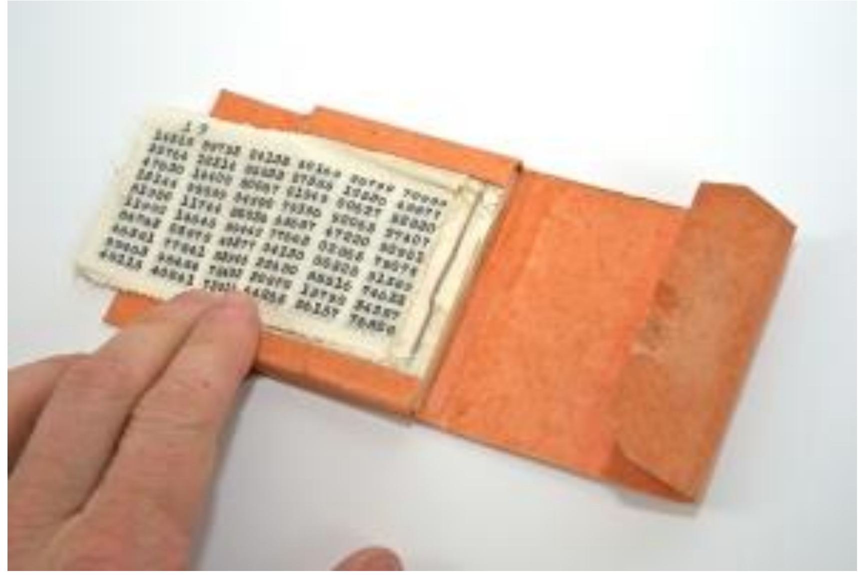
$$\begin{array}{rl} & K = \mathit{1001010} \\ \oplus & M = 0101101 \\ \hline & C = 1100111 \end{array}$$

$$\begin{array}{rl} & K = \mathit{1001010} \\ \oplus & C = 1100111 \\ \hline & M = 0101101 \end{array}$$

One-time pad

- KGB agents and controllers
  - E.g., Colonel Rudolf Abel, active in NYC, 1950s
- Called "one-time pad" because…
- Hotlines between Moscow and Washington D.C., Canberra and Moscow, etc.
  - U.S.-Moscow line created in1963 after Cuban missile crisis
  - Teleprinters with one-time tape system
  - Keying tapes delivered via embassies
  - Canberra-Moscow broken because Soviets reused Moscow-D.C. pad!

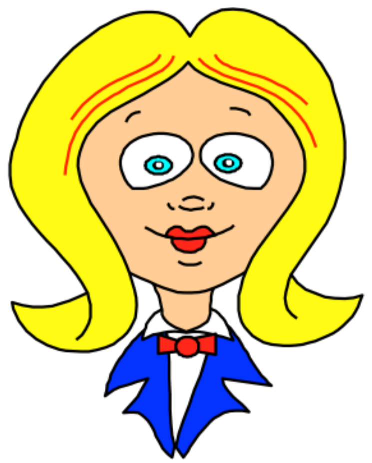




Unbreakable, but…

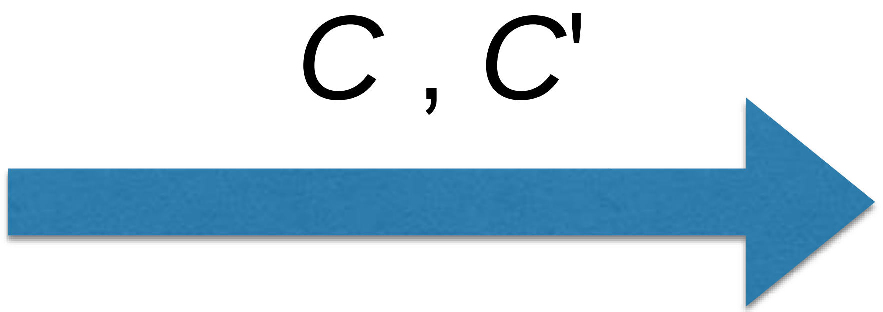
- One-time pad is one-time
  - Breakable if used twice

One-time pad—reloaded



$$\begin{array}{r} K = 1001010 \\ \oplus\; M = 0101101 \\ \hline C = 1100111 \end{array}$$

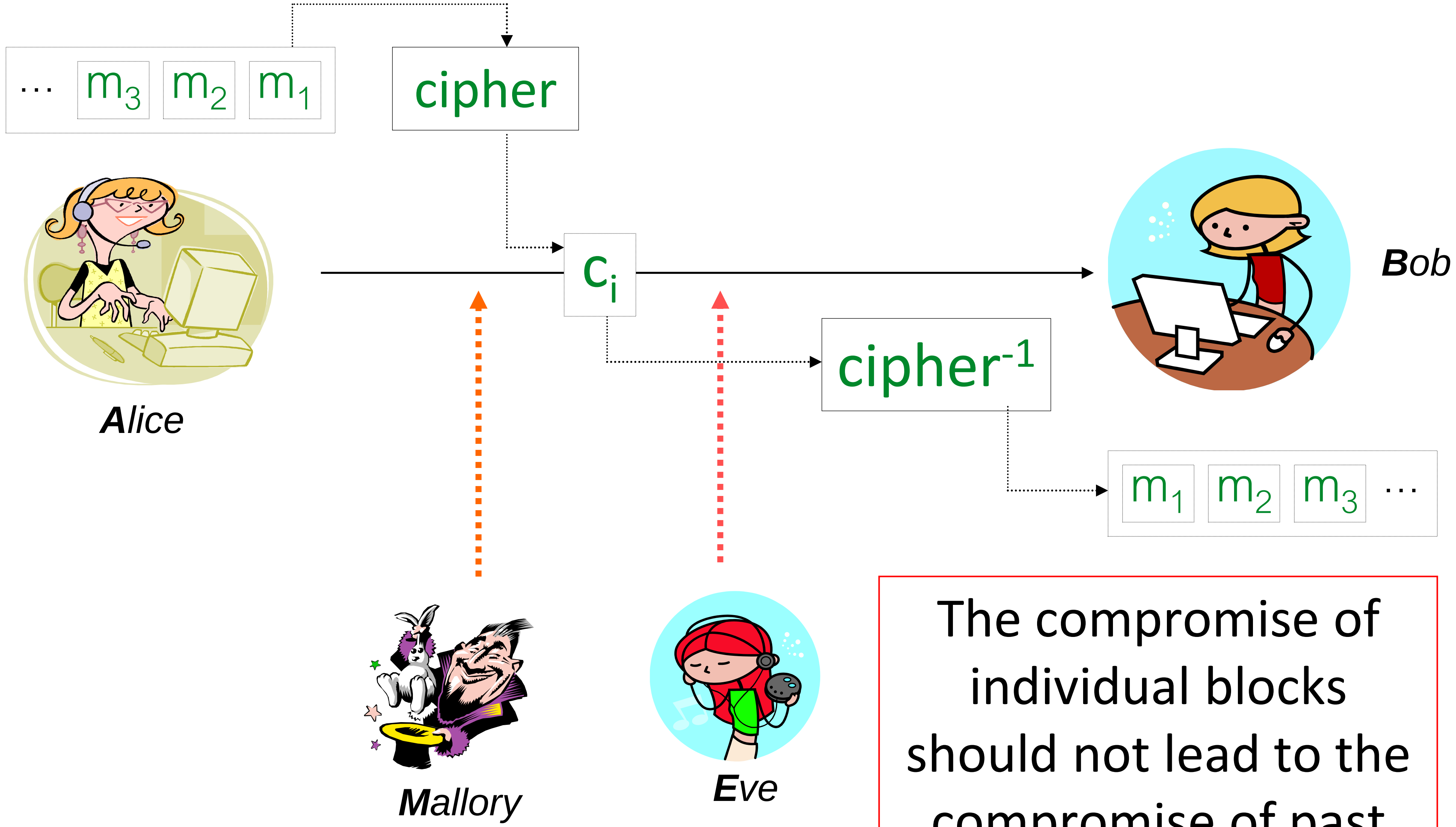
$$\begin{array}{r} K = 1001010 \\ \oplus\; M' = 010110\mathbf{0} \\ \hline C' = 110011\mathbf{0} \end{array}$$



Unbreakable, but…

- One-time pad is one-time
  - Breakable if used twice
- Key must be perfectly random
  - Randomness is a scarce resource
- Key length = message length very cumbersome!
- E.g., how can Alice encrypt her laptop hard drive?
  - Alice carries around hard drive containing the key?

Overview



The compromise of individual blocks should not lead to the compromise of past communication !

Challenges

- Using a cipher requires knowledge of threats in the environment in which it will be used
  - Is the set of possible messages small?
  - Do the messages exhibit regularities that remain after en-cipherment?
  - Can an active wire-tapper rearrange or change parts of the message?

Birthday paradox

- With 23 people in the same room chance of same birthday is over 50% !!!
- For **N** possible values expect a collision after seeing approx. **sqrt(N)** of them
- If **$N=2^n$** (**n**-bit key) after **$2^{n/2}$** (“birthday bound”) messages a collision is expected !

“Birthday attack” in action

- For 64-bit key, after seeing 2^{32} transactions Eve can find message sent with same key ! (how can she know ? Using keyed MAC of standard message header ?)

- Eve can then substitute old messages for new ones (e.g., reversing money transfers)

“meet in the middle” attack

- aka. “collision attack”
- Cousin of Birthday Attack

- $C = E_{K2}(E_{K1}(M))$
- This does not have 2n bit security !
- Why ?
- To find out whether C is an encryption of M:
  - T: Build table $E_K(M)$ for all K
  - Compute $D_K(C)$ for all K and lookup in T
  - Takes 2^{n+1} steps only

“pre-computation” attack

- If set of possible messages *M* is small
- Public key cipher *f* used
- Idea: pre-compute set of possible cipher-texts *f*(*M*), build table (*m*, *f*(*m*))
- When cipher-text *f*(*m*) appears, use table to find *m*
- Also called *forward searches*

Pre-computation in action

- Cathy knows Alice will send Bob one of two enciphered messages: BUY or SELL
- Using $public_B$, Cathy pre-computes

  $m_1 = E_{public_B}(\text{"BUY"})$

  $m_2 = E_{public_B}(\text{"SELL"})$
- Cathy sees Alice send Bob m_2
- Cathy knows Alice sent SELL

Fun non-obvious example

- Digitized sound
  - –Seems like far too many possible plaintexts
    - Initial calculations suggest 2^{32} such plaintexts
  - –Analysis of redundancy in human speech reduced this to about 100,000 ($\approx 2^{17}$)
    - small enough to worry about pre-computation attacks

Issue: mis-ordered blocks

- Alice sends Bob message
  - Message is LIVE (11 08 21 04)
  - Enciphered message is 44 57 21 16
- Eve intercepts it, rearranges blocks
  - Now enciphered message is 16 21 57 44
- Bob gets enciphered message, deciphers it
  - He sees EVIL

Handling mis-ordered blocks

- Signing each block won't stop it !
- Two approaches:
  - Crypto-hash the *entire* message and sign it
  - Place sequence numbers in each block of message, so recipient can tell intended order, then sign each block

More issues

- If plaintext repeats, ciphertext may too
- Example using DES:
  - input (in hex):

    3231 3433 3635 3837 **3231** 3433 3635 3837
  - corresponding output (in hex):

    ef7c 4bb2 b4ce 6f3b **ef7c** 4bb2 b4ce 6f3b
- Fix: cascade blocks together (chaining)
  - More details later

So what is going on then?

- Use of strong cryptosystems, well-chosen (or random) keys not enough to be secure
- Other factors:
  - Protocols directing use of cryptosystems
  - Ancillary information added by protocols
  - Implementation (not discussed here)
  - Maintenance and operation (not discussed here)

Stream ciphers, block ciphers

- E encryption function
  - $E_k(b)$ encryption of message b with key k
  - In what follows, $m = b_1 b_2 \ldots$, each b_i of fixed length
- Block cipher
  - $E_k(m) = E_k(b_1)E_k(b_2) \ldots$
- Stream cipher
  - $k = k_1 k_2 \ldots$
  - $E_k(m) = E_{k1}(b_1)E_{k2}(b_2) \ldots$
  - If $k_1 k_2 \ldots$ repeats itself, cipher is *periodic* and the length of its period is one cycle of $k_1 k_2 \ldots$

Examples

- Vigenère cipher
  - b_i = 1 character, $k = k_1k_2 \ldots$ where k_i = 1 character
  - Each b_i enciphered using $k_{i \bmod \text{length}(k)}$
  - Stream cipher
- DES
  - b_i = 64 bits, k = 56 bits
  - Each b_i enciphered separately using k
  - Block cipher

Stream ciphers

- Often (try to) approximate one-time pad by xor'ing each bit of key with one bit of message
  - Example:

$$m = 00101$$
$$k = 10010$$
$$c = 10111$$

- But how to generate a good key?

Synchronous Stream Ciphers

- n-stage Linear Feedback Shift Register:
  - n bit register $r = r_0 \ldots r_{n-1}$
  - n bit "tap sequence" $t = t_0 \ldots t_{n-1}$
  - Use:
    - Use r_{n-1} as key bit
    - Compute $x = r_0 t_0 \oplus \ldots \oplus r_{n-1} t_{n-1}$
    - Shift r one bit to right, dropping r_{n-1}, x becomes r_0

Example

- 4-stage LFSR; t = 1001

r	k_i	*new bit computation*	*new r*
0010	0	01⊕00⊕10⊕01 = 0	0001
0001	1	01⊕00⊕00⊕11 = 1	1000
1000	0	11⊕00⊕00⊕01 = 1	1100
1100	0	11⊕10⊕00⊕01 = 1	1110
1110	0	11⊕10⊕10⊕01 = 1	1111
1111	1	11⊕10⊕10⊕11 = 0	0111
0111	0	01⊕10⊕10⊕11 = 1	1011

  – Key sequence has period of 15 (010001011101110)

Make it difficult for bad guy

- n-stage *Non-Linear* Feedback Shift Register:
  - n bit register $r = r_0 \ldots r_{n-1}$
  - Use:
    - Use r_{n-1} as key bit
    - Compute $x = f(r_0, \ldots, r_{n-1})$; f is any function
    - Shift r one bit to right, dropping r_{n-1}, x becomes r_0

  Note same operation as LFSR but more general bit replacement function

Example

- 4-stage NLFSR; $f(r_0, r_1, r_2, r_3) = (r_0 \& r_2) \mid r_3$

r	k_i	*new bit computation*	*new r*
1100	0	(1 & 0) | 0 = 0	0110
0110	0	(0 & 1) | 0 = 0	0011
0011	1	(0 & 1) | 1 = 1	1001
1001	1	(1 & 0) | 1 = 1	1100
1100	0	(1 & 0) | 0 = 0	0110
0110	0	(0 & 1) | 0 = 0	0011
0011	1	(0 & 1) | 1 = 1	1001

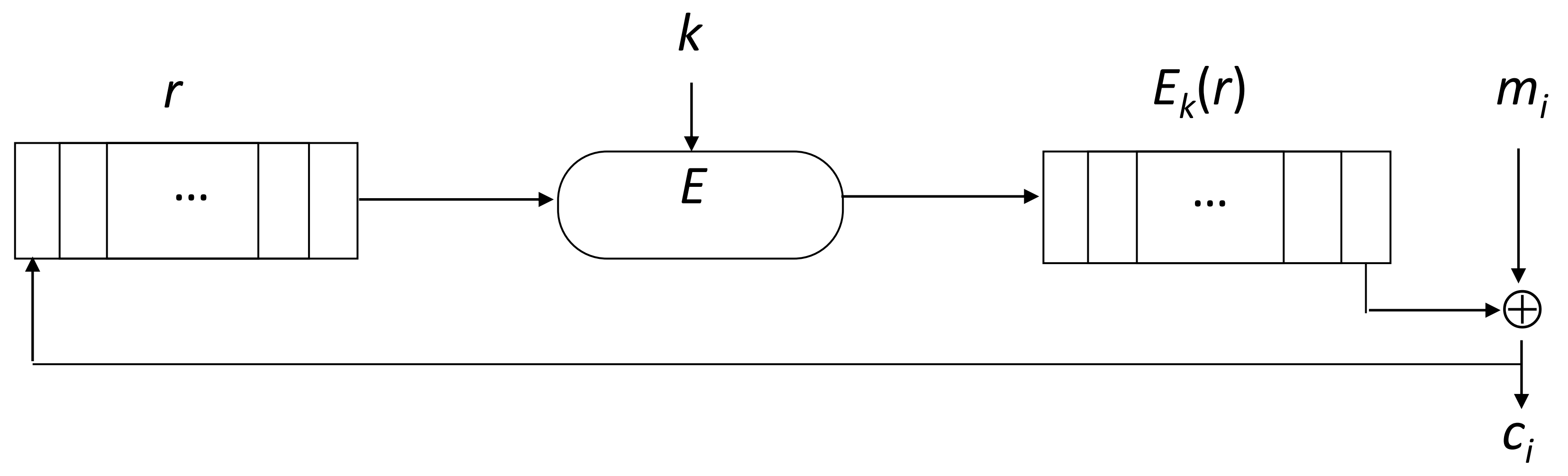
  – Key sequence has period of 4 (0011)

Making it even more difficult

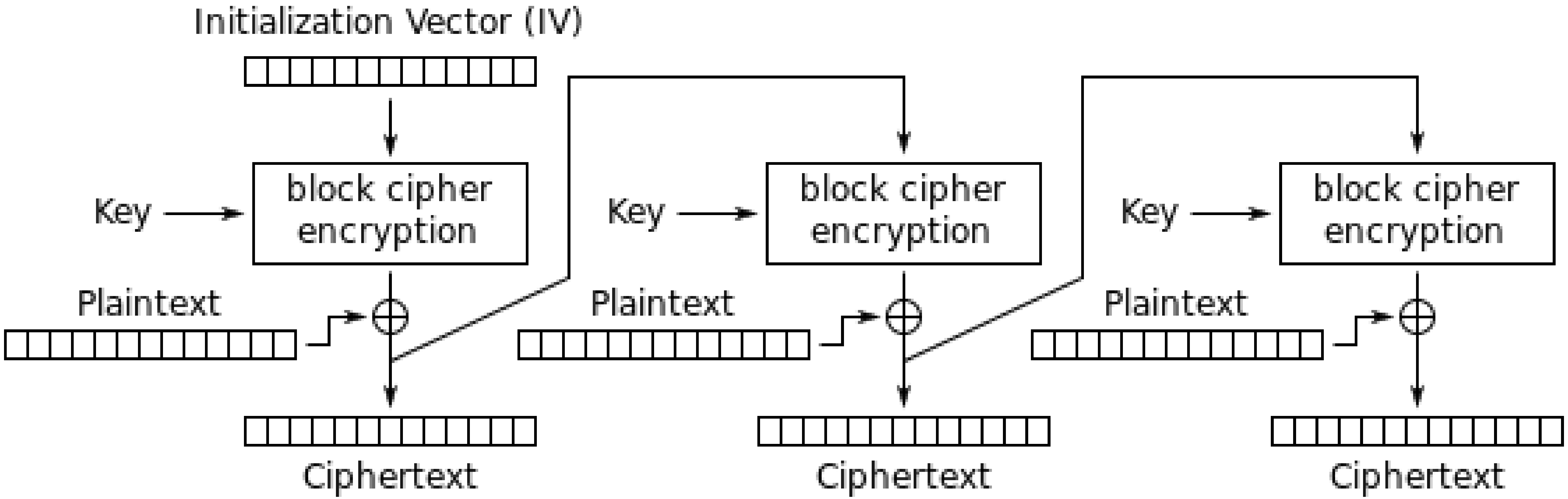
- NLFSRs not common
  - We don’t know how to design them to have long period
- Alternate approach: *output feedback mode*
  - For E encipherment function, k key, r register:
    - Compute $r' = E_k(r)$; key bit is rightmost bit of r'
    - Set r to r' and iterate, repeatedly enciphering register and extracting key bits, until message enciphered
  - Variant: use a counter that is incremented for each encipherment rather than a register
    - Take rightmost bit of $E_k(i)$, where i is number of encipherment

Cipher Feedback Mode (CFB)

- Cipher feedback mode: 1 bit of ciphertext fed into *n* bit register
  - Self-healing property: if ciphertext bit received incorrectly, it and next *n* bits decipher incorrectly; but after that, the ciphertext bits decipher correctly
  - Need to know *k*, *E* to decipher ciphertext



CFB



Cipher Feedback (CFB) mode encryption

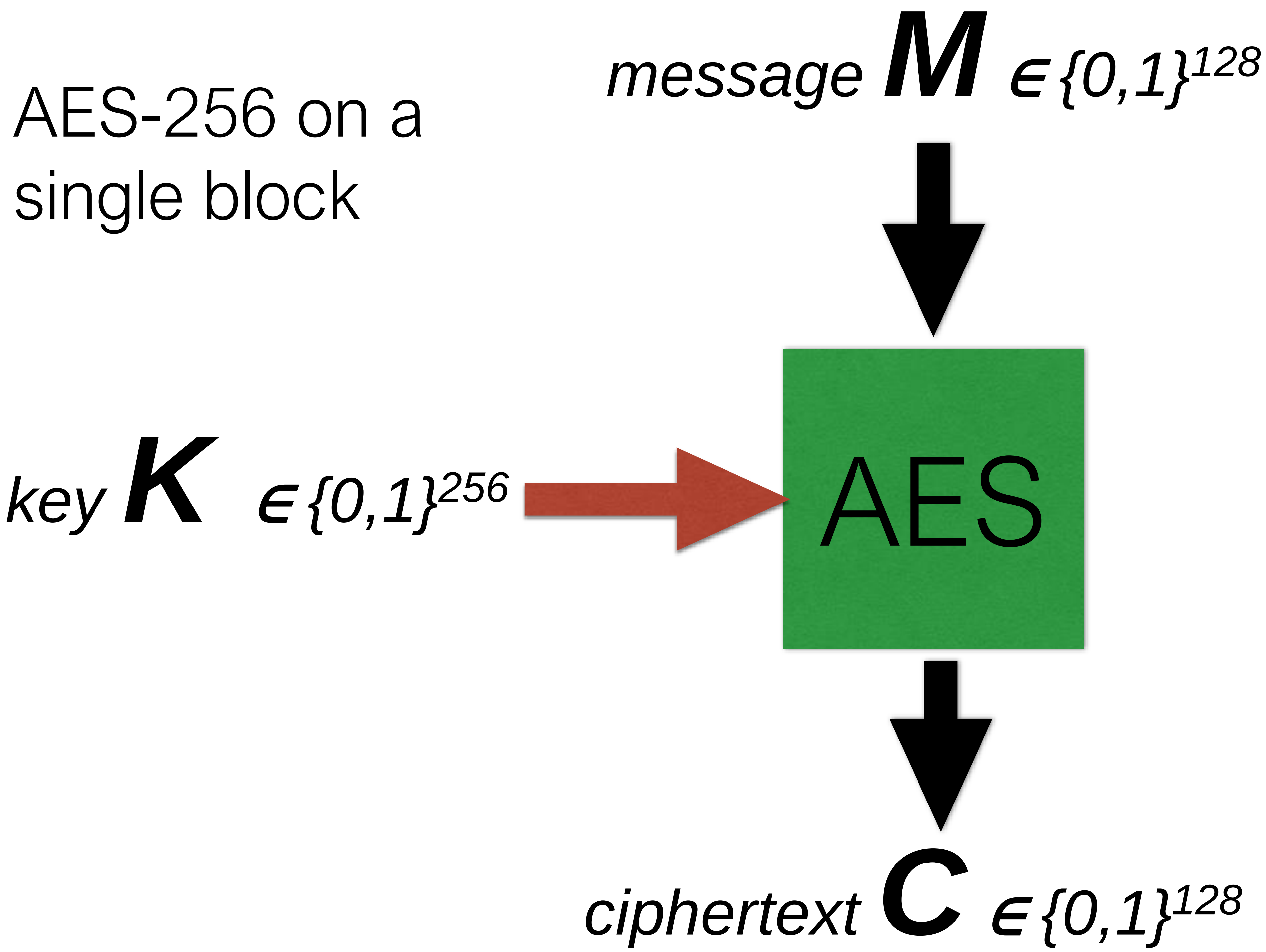
Block Ciphers

- Encipher, decipher multiple bits at once
- Each block enciphered independently
- Problem: identical plaintext blocks produce identical ciphertext blocks
  - Example: two database records
    - `MEMBER: HOLLY INCOME $100,000`
    - `MEMBER: HEIDI INCOME $100,000`
  - Encipherment:
    - `ABCQZRME GHQMRSIB CTXUVYSS RMGRPFQN`
    - `ABCQZRME ORMPABRZ CTXUVYSS RMGRPFQN`

Block cipher

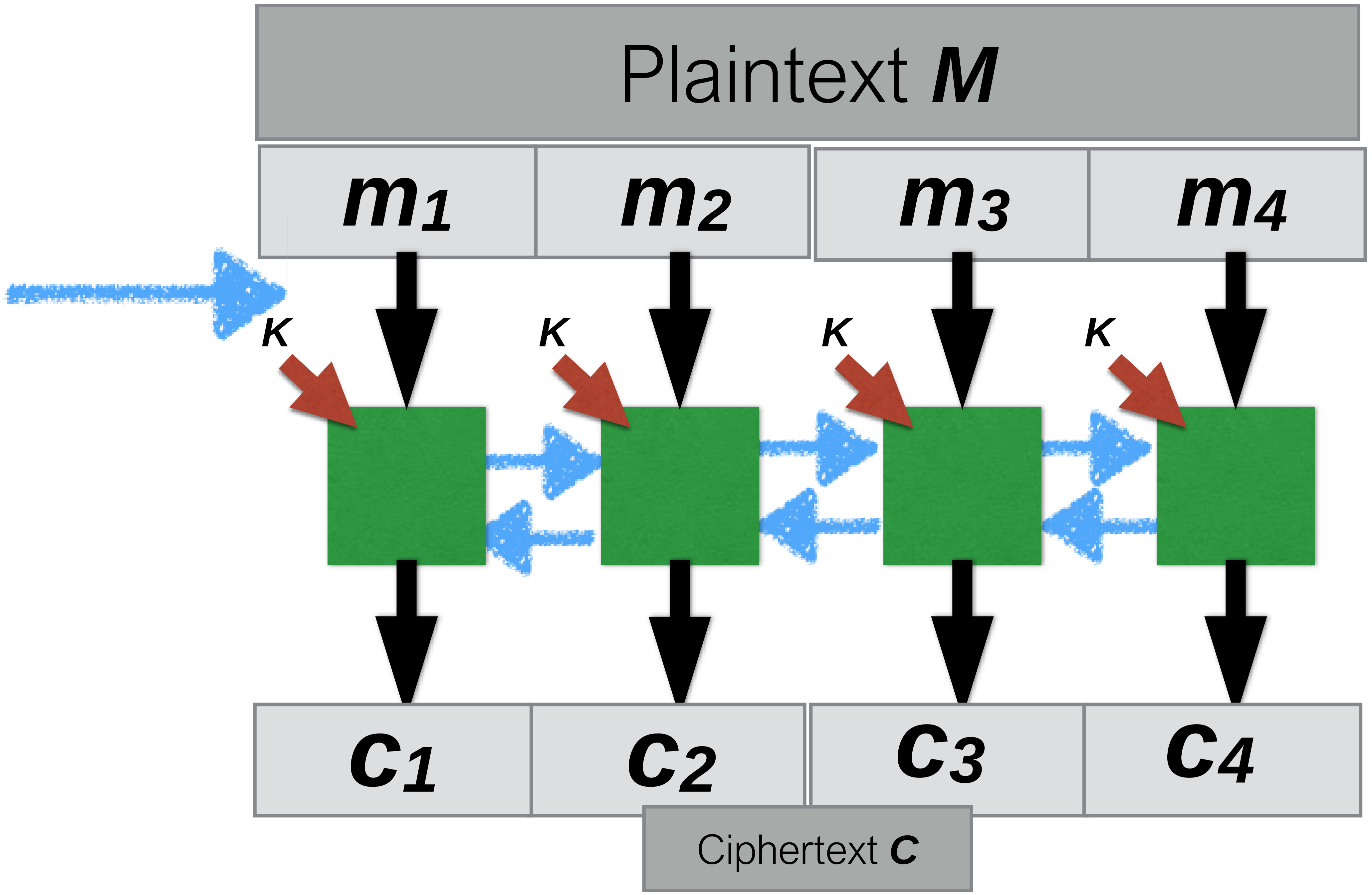
E.g., Advanced Encryption Standard (AES)

AES-256 on a single block



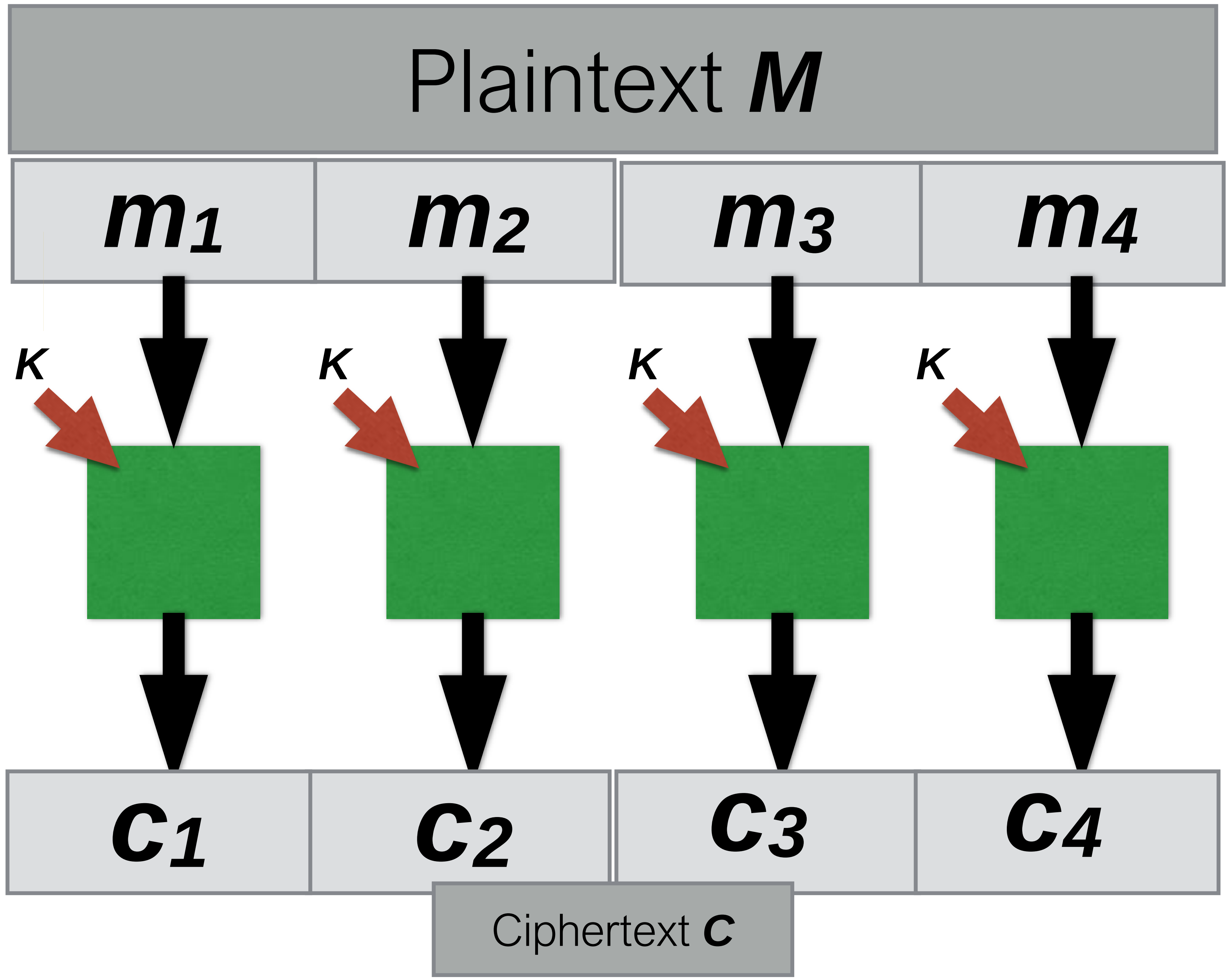
What if *M* is long?

Mode of operation



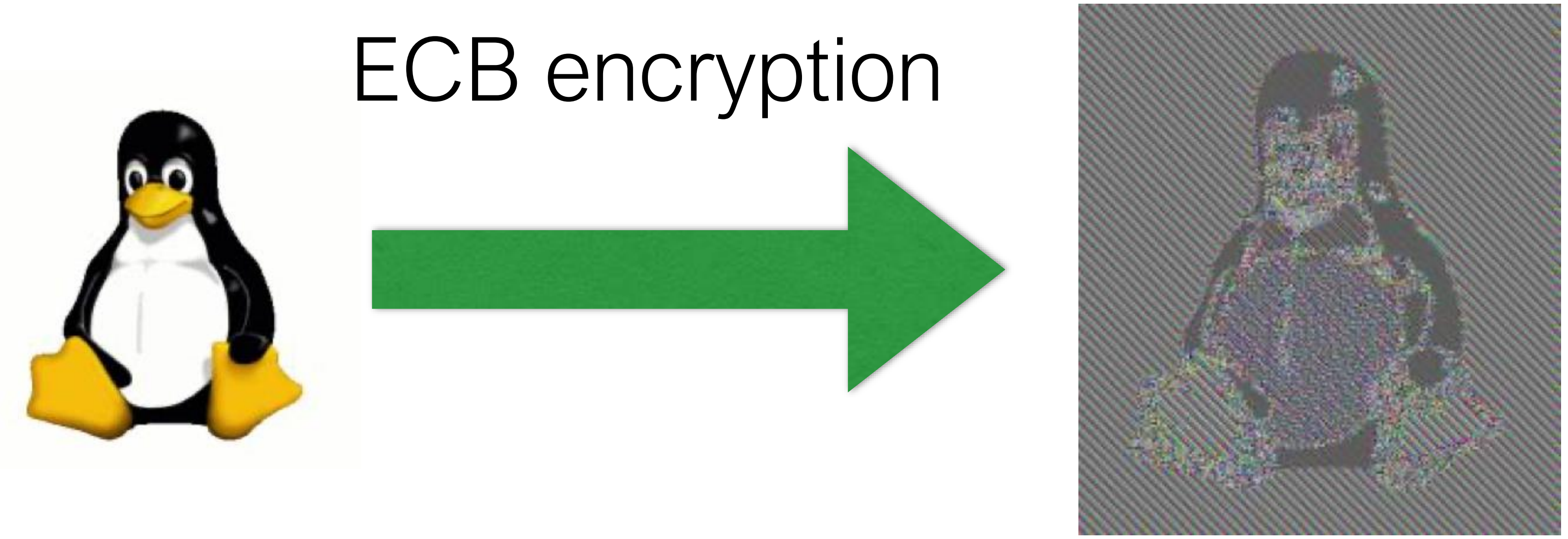
Various possible additions / interconnections:

Electronic Code Book (ECB) mode



Identical message blocks ➜ identical ciphertext blocks!

ECB leaks information



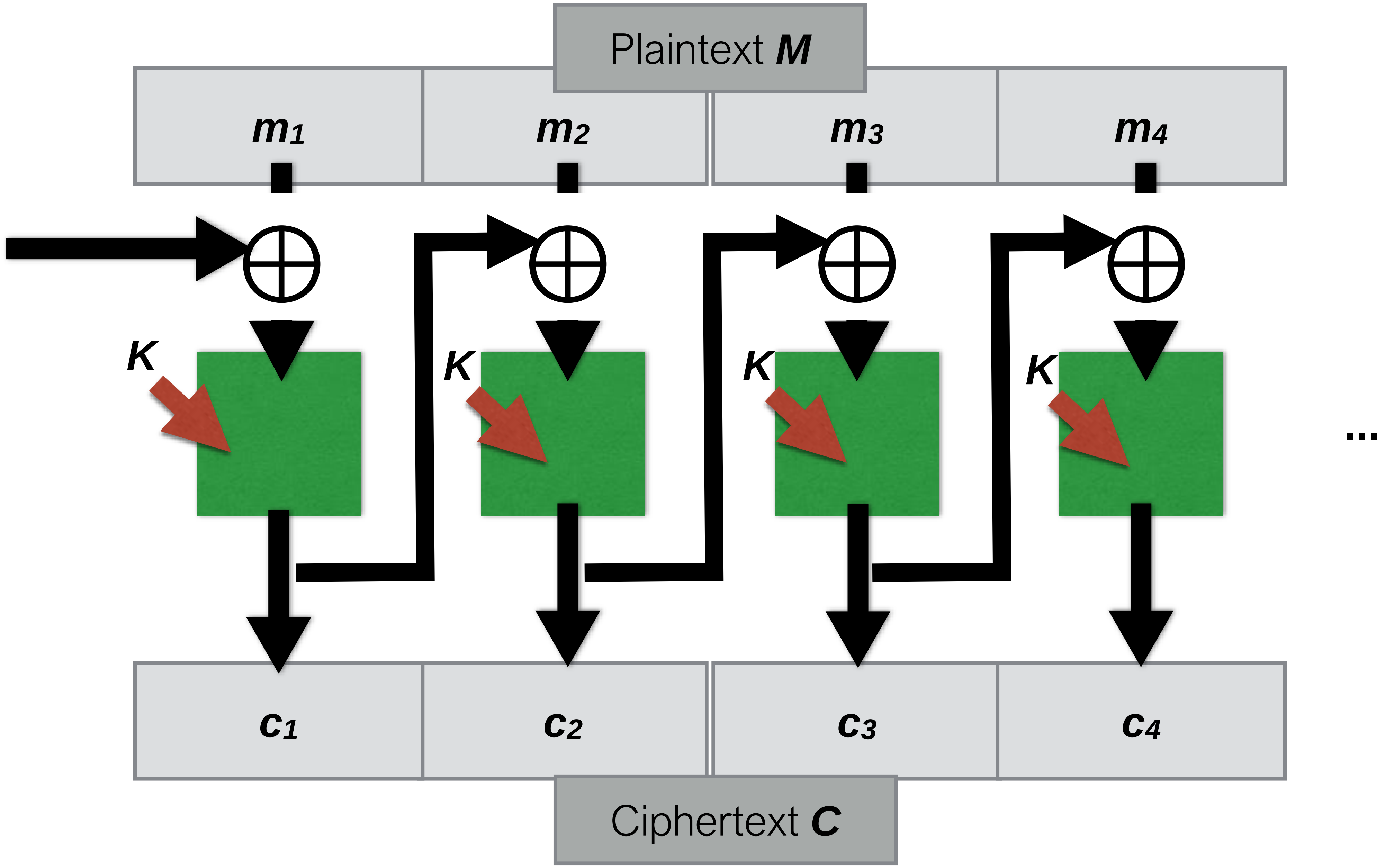
Idea

- Insert information about block's position into the plaintext block, then encipher.
- *Cipher block chaining mode (CBC)*:
  - Exclusive-or current plaintext block with previous ciphertext block:
    - $c_0 = E_k(m_0 \oplus I)$
    - $c_i = E_k(m_i \oplus c_{i-1})$ for i > 0

    where I is the initialization vector

Cipher-Block Chaining (CBC) mode

Fresh(!), random initialization vector (IV)



- Identical message blocks now encrypted differently
- Approach similar to Merkle-Damgard

Issue with chaining

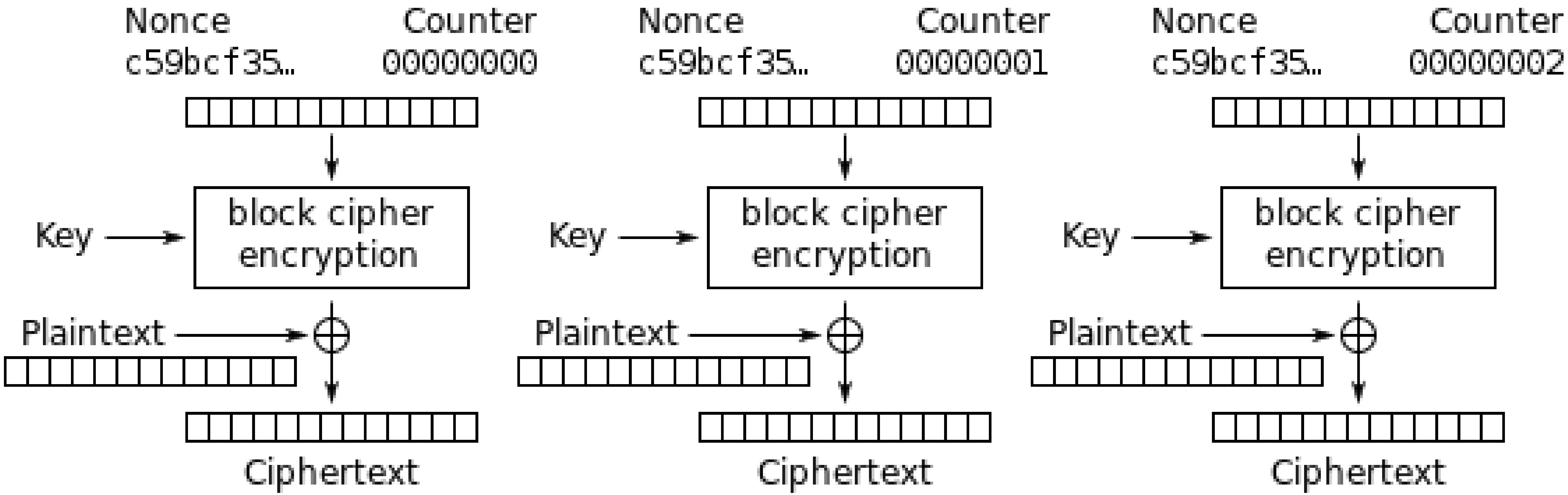
How do we access/decrypt random blocks without having to decrypt everything “before”?

Solution: CTR

- *Counter mode (CTR)*:
  - –Key constructed by encrypting block counter
    - $k_i = E_k(unique\_nonce||i)$
    - $c_i = m_i \oplus k_i$

    e.g. unique_nonce=(message number)
  - –Question: why do we need the *nonce* ?
  - –Careful: never use same (*k*,*nonce*) pair !!!

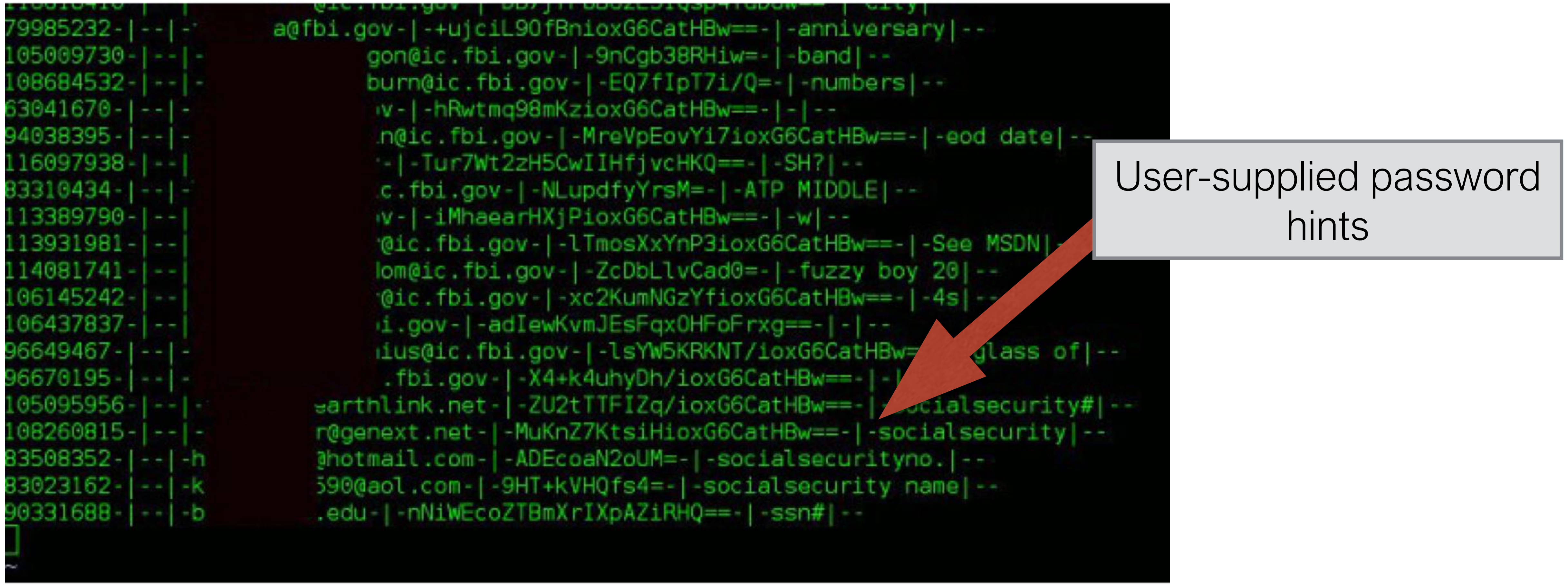
CTR

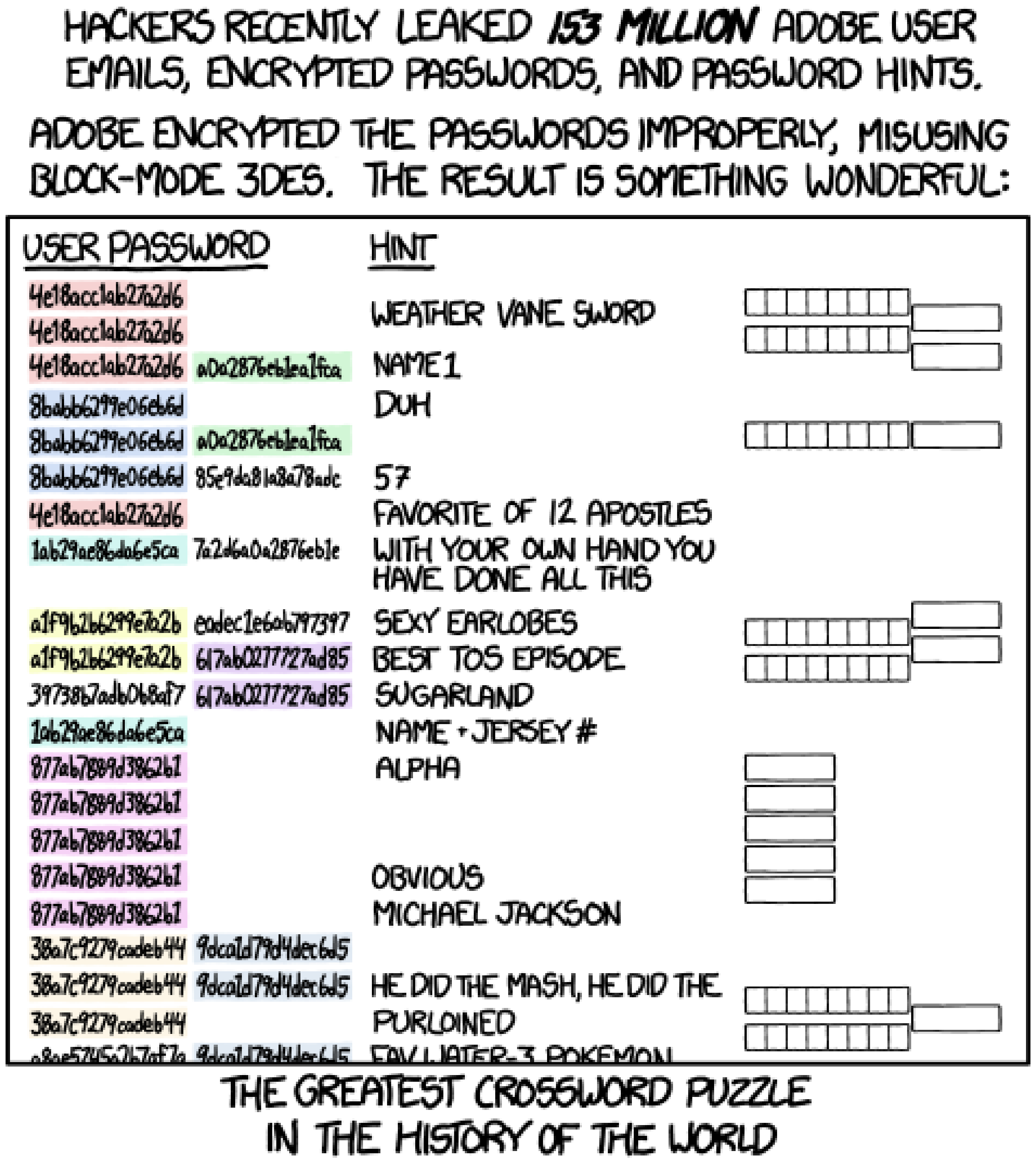


Counter (CTR) mode encryption

What if we choose the wrong mode?

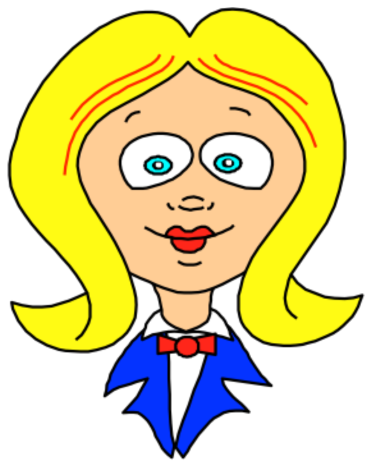
- Adobe breach leaked 153 million passwords in 2013
- Encrypted using ECB, not hashed with salt
    - Key remained secret, but…





xkcd on the Adobe breach

Integrity problem



$K = 1001010$

$\oplus\ M = 0101101$

$C = 1100111$

$C \Rightarrow C'$

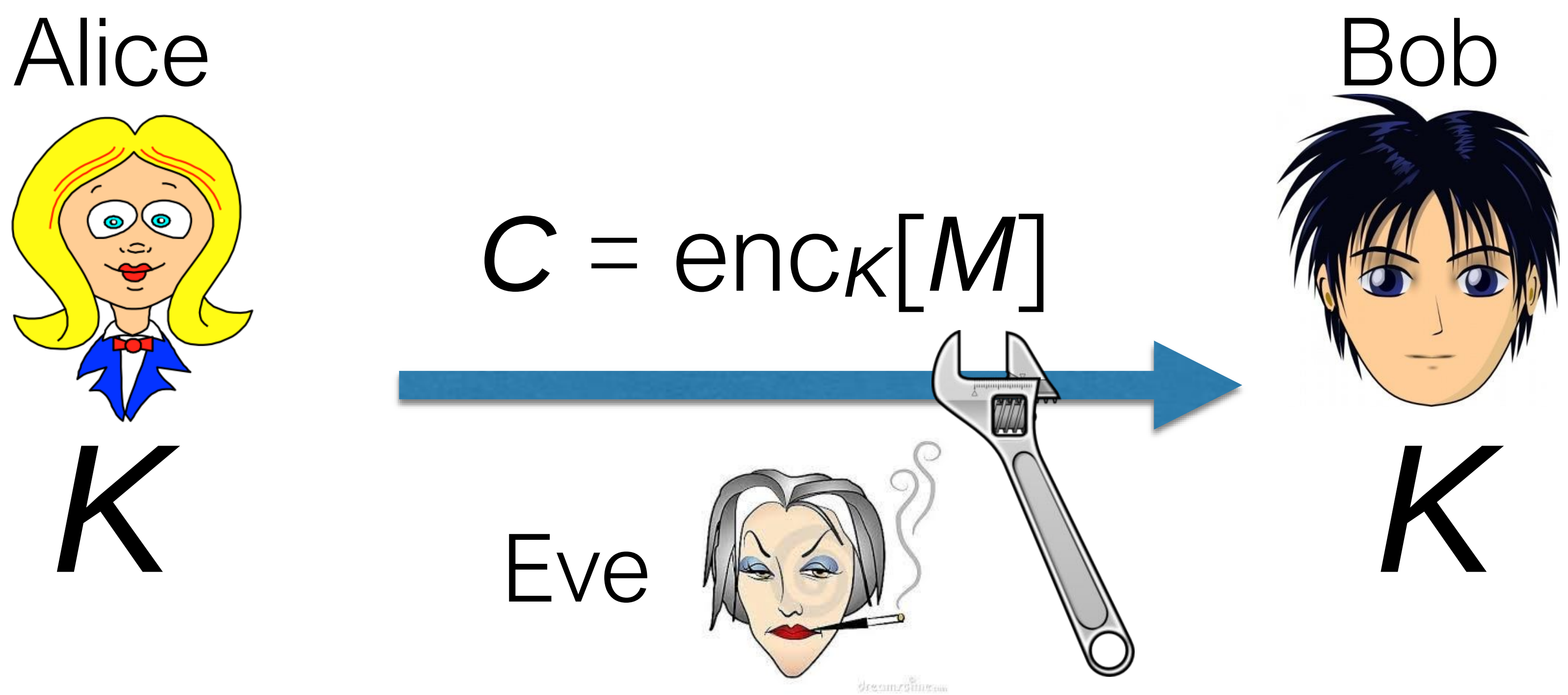
$M' = 010110\mathbf{0}$

$C' = 110011\mathbf{0}$

Eve

What about integrity?

- Also want Eve not to modify *C* (and potentially *M*) without detection
- ***Authenticated*** encryption modes (e.g., OCB) ensure such integrity.
- Can also use a ***message authentication code*** (MAC)
  - E.g., HMAC (Bellare, Canetti, Krawczyk 1996), uses hash function
  - Encrypt + MAC



Kerckhoffs's Principle

- *"The design of a [crypto]system should not require secrecy…"*
- Counterintuitive!
- Encryption should be secure even if the adversary (Eve) knows the algorithm ***enc***.
- Thus:
  - Security relies on *secrecy of key **K***
  - Key ***K*** must be *random* and of adequate length (e.g., 128 bits)

Jean Guillaume **Auguste** Victor François Hubert **Kerckhoffs** (1835-1903)

In fact, *everyone* knows *enc*

- Advanced Encryption Standard (AES)
  - Published by NIST in 2001 after five-year contest (FIPS PUB 197)
  - Extremely wide use (TLS, NSA top secret, etc.)
  - Block cipher with 128, 192, and 256-bit key variants based on Rijndael cipher
  - 128-bit message blocks (as we've seen)
  - Very fast
    - 1500 Mbps with AES-NI on 2.4 GHz Intel Westmere (IPSec, 1kB packets, with hyperthreading, AES-128-GCM) [Source: 2010 Intel whitepaper 324238-001]
- There are other good ciphers, but AES dominates

Optional for next week

For **+0.5% credit**.

Install **openssl** and decrypt **any** of the following ciphertexts:

U2FsdGVkX18Avp0s9oaA8I2HeaLoCG1gZyRmoLWWBFZXcrm/1ZsXSjxc2XTpbPZw

U2FsdGVkX18KRUFApfRXdayMo8sYd96zEAdPXyA4hzMBdWxqVigJGsLs4okBhwje

U2FsdGVkX1/DUTj3FPMhUWb/hgxIchBN6LWoRbLm2L/CARN/VSAYlg==

U2FsdGVkX1/+vE2czERZciAIJteLkzndHwW9QrdibZ/Z6q8=