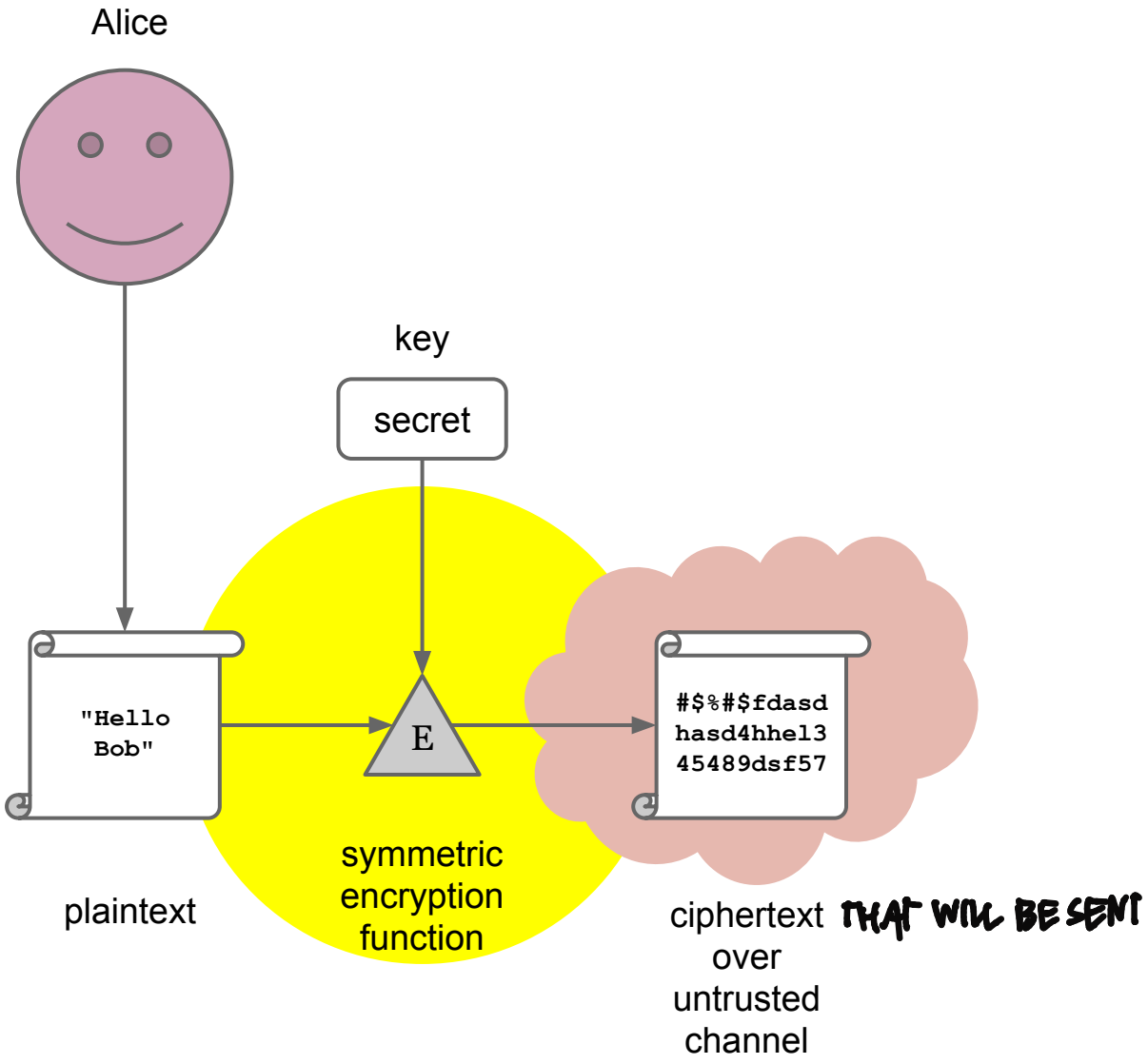
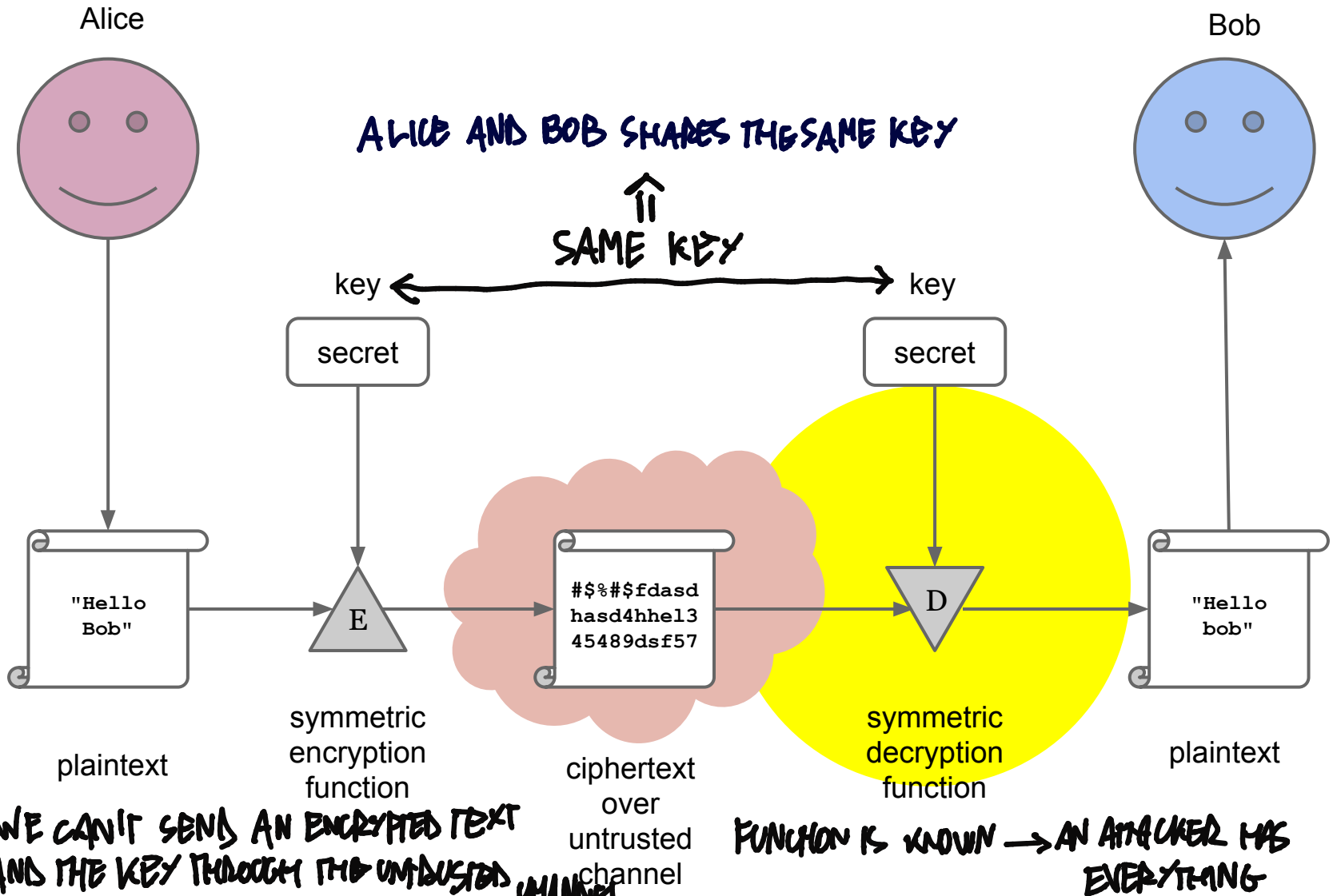
Symmetric Encryption

Confidentiality

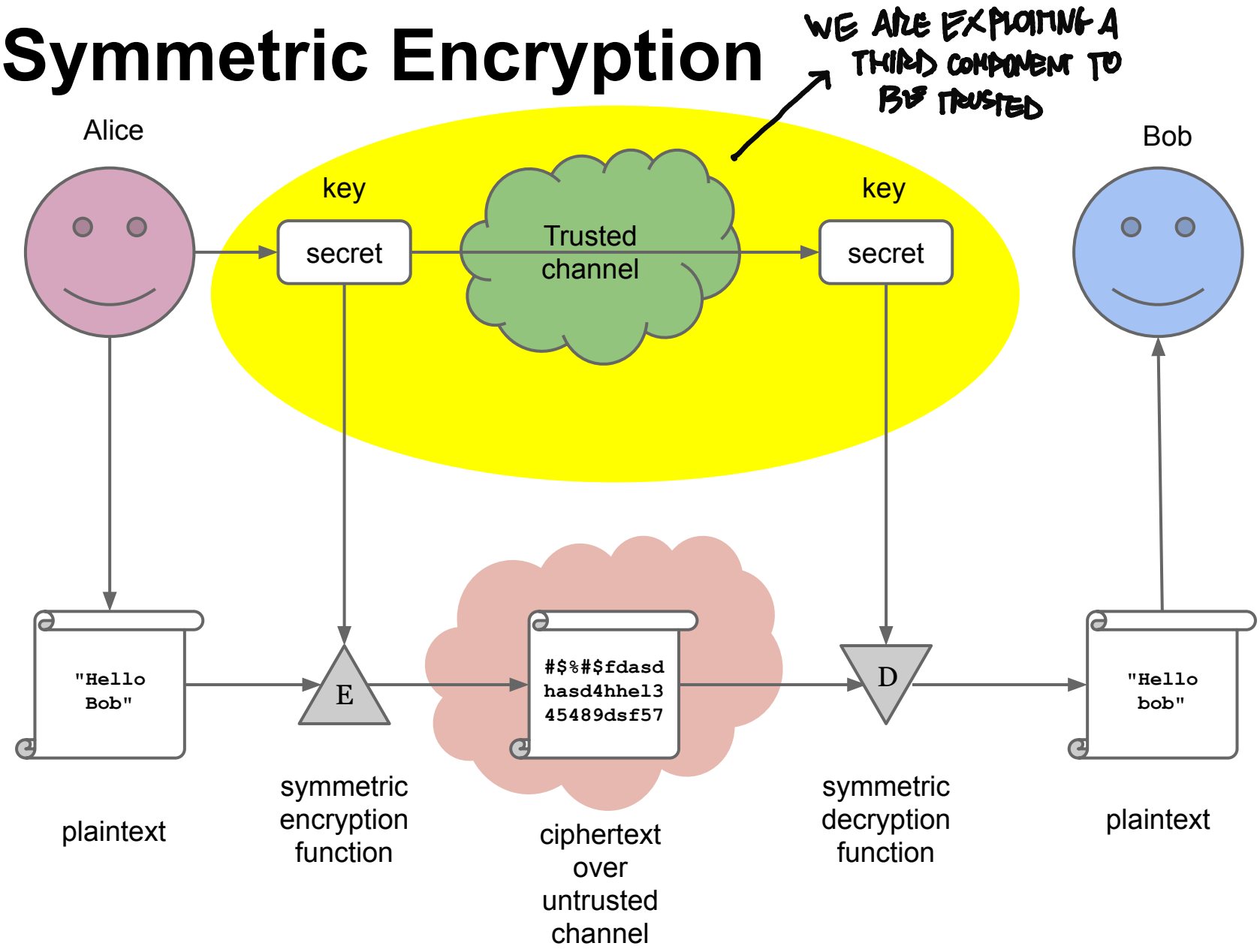
Symmetric Encryption



Symmetric Encryption



Symmetric Encryption



Symmetric Encryption

- The basic idea of encryption
  - Use key K to ***encrypt*** plaintext in ciphertext
  - Use same key K to ***decrypt*** ciphertext in plaintext
- Synonyms: shared key encryption, secret key encryption
- **Issue:** how do we agree on the key?
  - Cannot send key on same channel as message!
  - Off-band transmission mechanism needed
- **Issue:** scalability
- A symmetric algorithm is a cocktail...

First ingredient: substitution

Substitution: "replacing each byte with another"

Toy example (Caesar cipher)

- replace each letter in a sentence with the one following it by K positions in the alphabet
- Example: "`SECURE`" becomes "`VHFXUH`" with $K = 3$

Many issues (it's a toy example!)

First ingredient: substitution

Substitution: “replacing each byte with another”

Toy example (Caesar cipher)

- replace each letter in a sentence with the one following it by K positions in the alphabet
- Example: “**`SECURE`**” becomes “**`VHFXUH`**” with $K = 3$

Many issues (it’s a toy example!)**:**

- if cipher known, with 25 attempts at most, 13 on average, we have the key: **keyspace too small**

First ingredient: substitution

Substitution: “replacing each byte with another”

Toy example (Caesar cipher)

- replace each letter in a sentence with the one following it by K positions in the alphabet
- Example: “**`SECURE`**” becomes “**`VHFXUH`**” with $K = 3$

Many issues (it’s a toy example!):

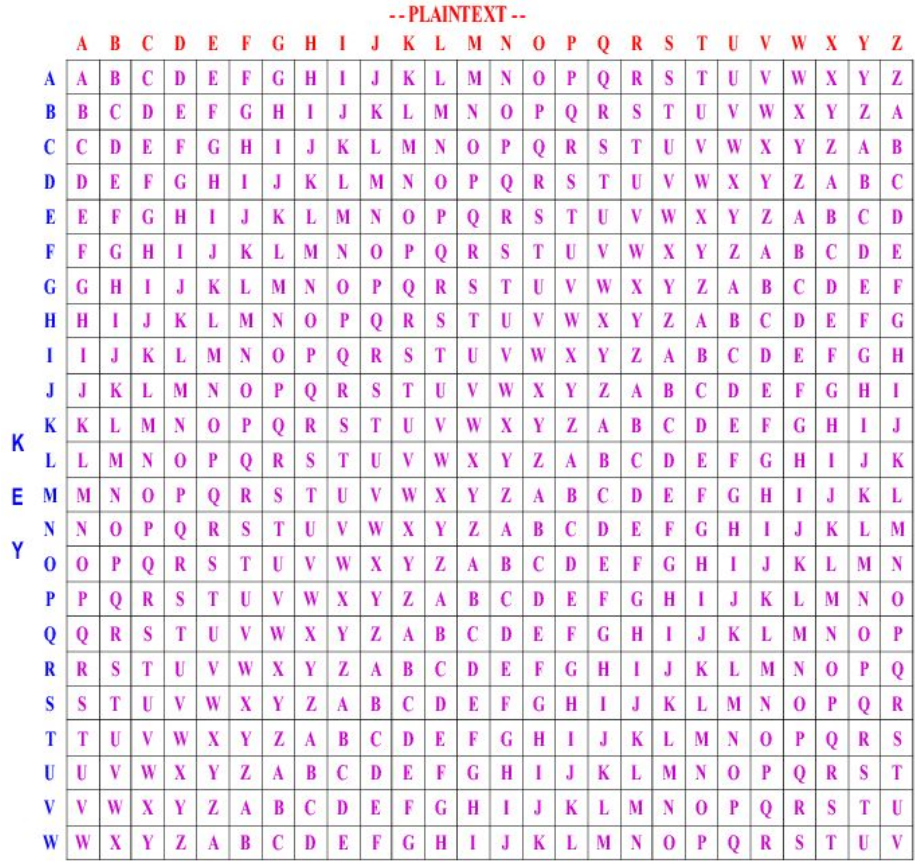
- if cipher known, with 25 attempts at most, 13 on average, we have the key: **keyspace too small**.
- **repetitions** and **structure** "visible" in ciphertext: monoalphabetic ciphers are weak (frequency analysis - CPTX only attack) https://en.wikipedia.org/wiki/Letter_frequency

Polyalphabetic ciphers (Vigenere Cipher)

m=SECURE

k=SECRET

c=?

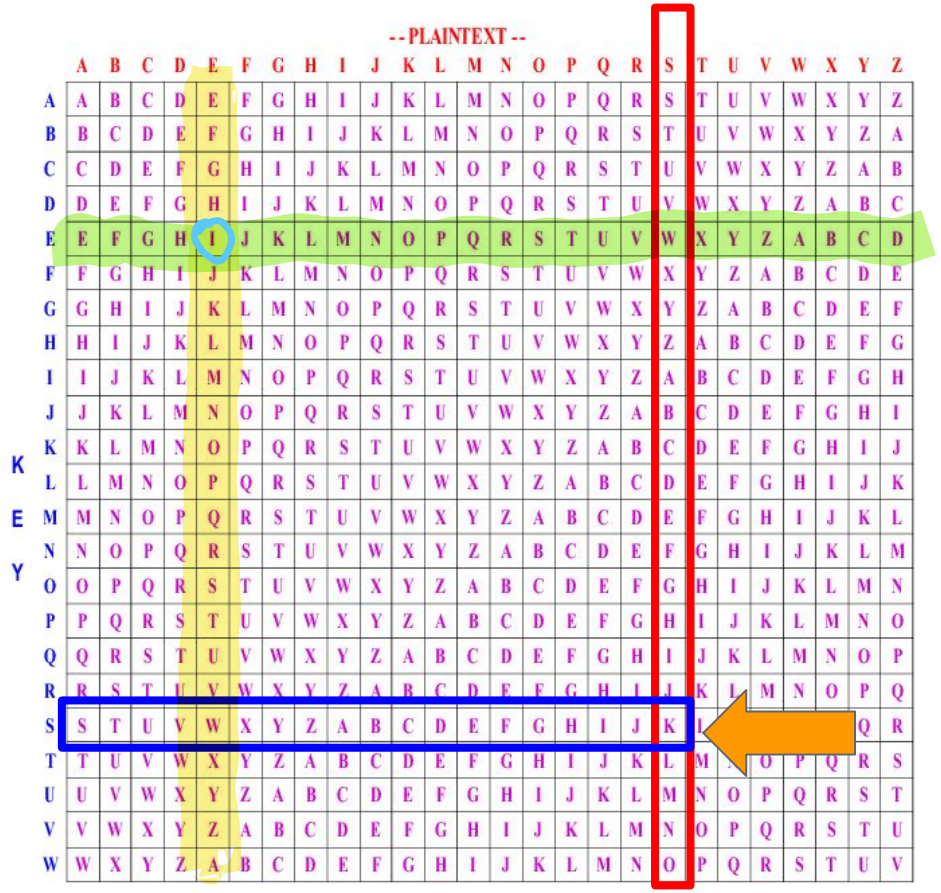


Polyalphabetic ciphers (Vigenere Cipher)

m=SECURE

k=SECRET

c=KI....

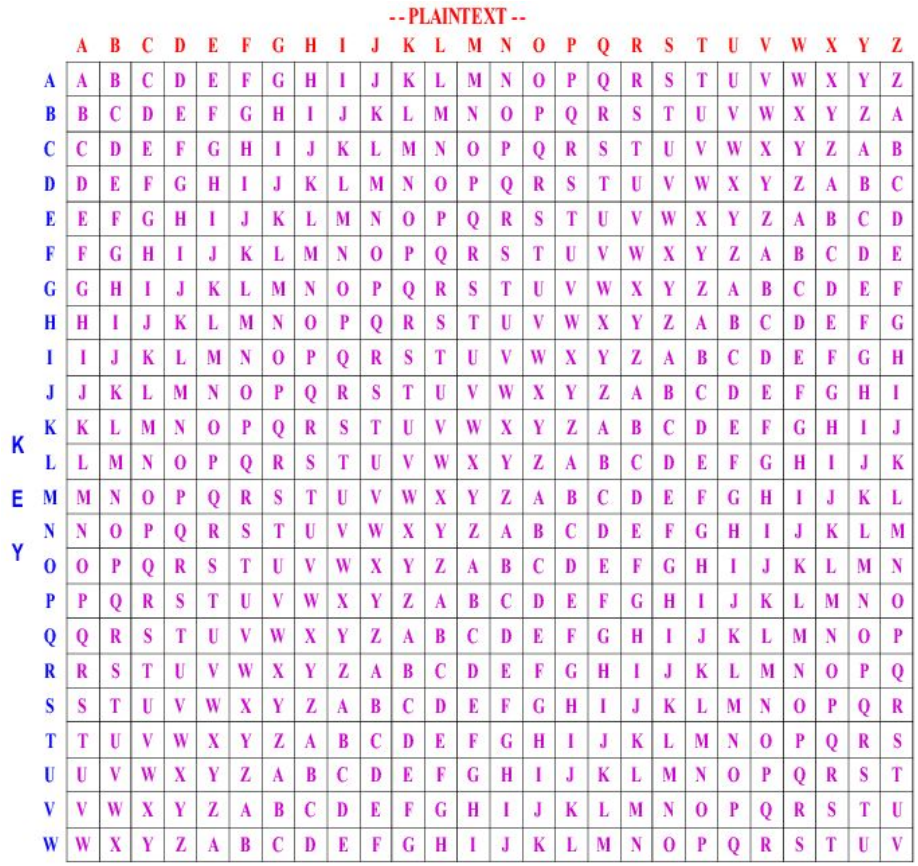


Polyalphabetic ciphers (Vigenere Cipher)

m=SECURE

k=SECRET

c=KIELVX

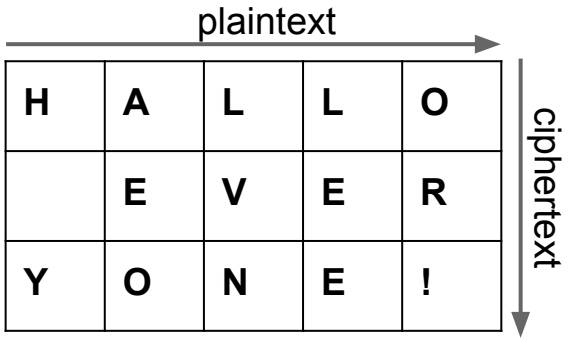


KEY \ --PLAINTEXT--	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V

ADVANTAGE W.R.T. MONOALPHABETIC:
REPETITIONS AND STRUCTURE PROBLEMS ARE GONE

Second ingredient: transposition

Transposition (or diffusion) means "swapping the values of given bits"



Toy example (matrix):

- Write by rows, read by columns
- Key: $K = (R, C)$ with $\mathrm{R} * \mathrm{C} \sim \mathrm{len}(\mathrm{msg})$

Many issues (it's a toy example!)

Example - Diffusion

$M_{3\times5}$ =

H	A	L	L	O
O	E	V	E	R
Y	O	N	E	!

ADVANTAGE: NO REPETITION

m= HALLO EVERYONE!

k=(3,5)

c=H YAEOLVNLEEOR!

Example - Diffusion

m= HALLO

k=(3,5)

c=H A L L O

H	A	L	L	O

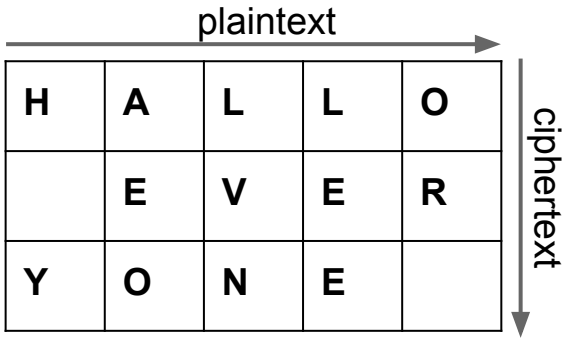
R * C >> len(msg) TO BE EFFICIENT

15 >> 4

OTHERWISE IT'S LIKE THERE IS NO ENCRYPTION

Second ingredient: transposition

Transposition (or diffusion) means “swapping the values of given bits”



Toy example (matrix):

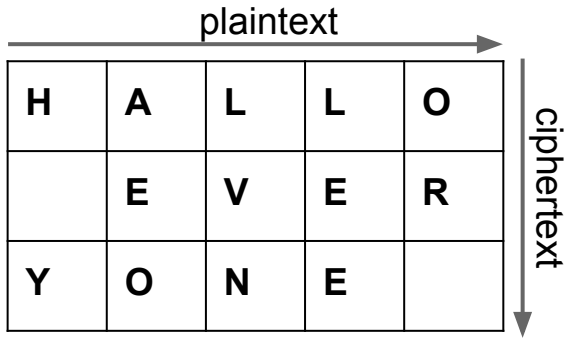
- Write by rows, read by columns
- Key: $K = (R, C)$ with $\mathrm{R} * \mathrm{C} \sim \mathrm{len}(\mathrm{msg})$

Many issues (it’s a toy example!):

- Keyspace still relatively small

Second ingredient: transposition

Transposition (or diffusion) means “swapping the values of given bits”



Toy example (matrix):

- Write by rows, read by columns
- Key: $K = (R, C)$ with $\mathrm{R} * \mathrm{C} \sim \mathrm{len}(\mathrm{msg})$

WHEN DESIGNING A MODERN CRYPTOGRAPHY, IT IS RECOMMENDED TO COMBINE SUBSTITUTION AND TRANSPOSITION TO PRODUCE A SECURE CIPHERTEXT

Many issues (it’s a toy example!):

- Keyspace still relatively small

But **repetitions** and **structure** gone

- We now really need to test all possible structures

Perfectly secure cipher

$\Pr \rightarrow$ PROBABILITY

Definition

- In a *perfect cipher*, for all $\texttt{ptx} \in \mathbf{P}$ and $\texttt{ctx} \in \mathbf{C}$,
  $\Pr(\text{ptx sent} = \texttt{ptx}) = \Pr(\text{ptx sent} = \texttt{ptx} \mid \text{ctx sent} = \texttt{ctx})$
- In other words: seeing a ciphertext $c \in \mathbf{C}$ gives us *no information* on what the plaintext corresponding to c could be

Question

- The definition is not constructive! Does a perfect cipher exist?
  - If yes, what does it look like?

Perfectly secure cipher

IF YOU WANT TO ENCRYPT n PLAINTEXTS, YOU NEED n KEYS AND n CIPHERTEXTS

Theorem (Shannon 1949)

Any symmetric cipher $\langle \mathbf{P}, \mathbf{K}, \mathbf{C}, \mathbb{E}, \mathbb{D} \rangle$ with $|\mathbf{P}| = |\mathbf{K}| = |\mathbf{C}|$ is perfectly secure if and only if

- every key is used with probability $\frac{1}{|\mathbf{K}|}$ — THE EXTRACTED KEY IS RANDOMIC
- a unique key maps a given plaintext into a given ciphertext: $\forall(\mathtt{ptx}, \mathtt{ctx}) \in \mathbf{P} \times \mathbf{C},\ \exists! k \in \mathbf{K} \text{ s.t. } \mathbb{E}(\mathtt{ptx}, k) = \mathtt{ctx}$ — DIFFERENT KEYS FOR DIFFERENT PLAINTEXTS AND RESULTING IN DIFFERENT CIPHERTEXTS

A PERFECT CIPHERTEXT PRESERVES CONFIDENTIALITY BY DEFINITION

→ EVERY SINGLE TIME WE WANT TO ENCRYPT A TEXT, WE EXTRACT RANDOMICALLY A KEY AND IT WILL BE USED FOR THAT PLAINTEXT ONLY

Perfectly secure cipher

Theorem (Shannon 1949)

Any symmetric cipher $\langle \mathbf{P}, \mathbf{K}, \mathbf{C}, \mathbb{E}, \mathbb{D} \rangle$ with $|\mathbf{P}| = |\mathbf{K}| = |\mathbf{C}|$ is perfectly secure if and only if

- every key is used with probability $\frac{1}{|\mathbf{K}|}$
- a unique key maps a given plaintext into a given ciphertext: $\forall(\mathtt{ptx}, \mathtt{ctx}) \in \mathbf{P} \times \mathbf{C},\ \exists! k \in \mathbf{K}$ s.t. $\mathbb{E}(\mathtt{ptx}, k) = \mathtt{ctx}$

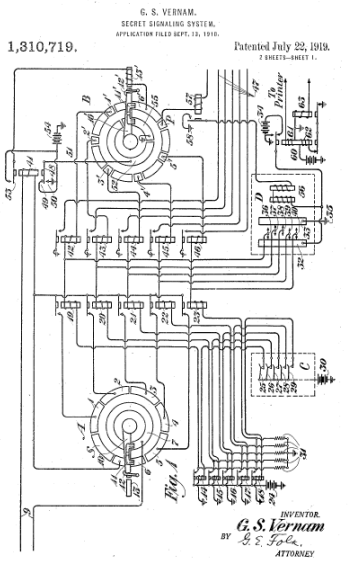
A simple working example

- Assume $\mathbf{P}, \mathbf{K}, \mathbf{C}$ to be set of binary strings. The encryption function draws a *uniformly random, fresh* key k out of $\mathbf{K}$ each time it is called and computes $\mathtt{ctx} = \mathtt{ptx} \oplus k$

Anticausal implementations

A concrete apparatus

- Gilbert Vernam actually patented a telegraphic machine implementing $\texttt{ptx} \oplus k$ on Baudot code in 1919
- Joseph Mauborgne suggested the use of a random tape containing k
- Using Vernam's encrypting machine with Mauborgne's suggestion implements a perfect cipher


G. S. VERNAM.
SECRET SIGNALING SYSTEM.
APPLICATION FILED SEPT. 13, 1918.
1,310,719.
Patented July 22, 1919.
2 SHEETS—SHEET 1.
Printer
Fig. 1
INVENTOR.
G. S. Vernam
BY
ATTORNEY

(ALSO KNOWN AS VERNAN CIPHER)

The One Time Pad: Perfect Cipher

EXAMPLE OF A PERFECT CIPHER THAT HAS BEEN USED IN THE PAST

- XOR of a message m and a random key k of the same size of m: $len(k) = len(m)$
  - The key is pre-shared and consumed while writing. Can never be re-used again!
- The OTP is a **minimal perfect cipher**
  - Minimal because $|K| = |M|$ GIVEN ANY CIPHERTEXT, AND ANY PLAINTEXT OF THE SAME LENGHT, THERE'S A KEY THAT DECRYPTS THE CIPHERTEXT TO THE PLAINTEXT

```
Plain text: T H I S  I S  S E C R E T
OTP-Key   : X V H E  U W  N O P G D Z
-------------------------------------
Ciphertext: Q C P W  C O  F S R X H S
In groups : QCPWC OFSRX HS
```

Perfectly secure $\neq$ perfectly usable

Key storage/management

- storing key material and changing keys is a nightmare
- perfect cipher broken in practice due to key theft/reuse
- generating random keys was also an issue (and caused breaks)

Photo courtesy of Cryptomuseum.com

Imperfections and Brute Force

- Real-world algorithms are not perfect (|K|<|M|), and so can be broken
  - each ciphertext-plaintext pair leaks a small amount of information (because the key is re-used)
- Only thing unknown is the key (Kerckhoffs)
  - Remember: the algorithm itself is known!
- Brute forcing is possible for any real world cipher
  - Try all possible keys, until **one** produces an output that "*makes sense*".
- **Perfect ciphers (one time pads) are not vulnerable to Brute Force**
  - because trying all the (random) keys will yield all the (possible) plaintexts, which are all equally likely (= no clue)

AND USUALLY KEYS ARE SMALLER THAN THE MESSAGE TO SEND

Non Perfect Cipher Example

M = UGO (21 7 15)

C = XJR (24 10 18)

Bruteforcing: k=-1-2-3.....-26 For each k, he/she shifts all letters...

Non Perfect Cipher Example

M = UGO (21 7 15)

K=+3

C = XJR (24 10 18)

Bruteforcing: k=-1-2-3.....-26 For each k, he/she shifts all letters...

K=-1 ... M1 = WIQ

Non Perfect Cipher Example

M = UGO (21 7 15)

K=+3

C = XJR (24 10 18)

Bruteforcing: k=-1-2-3.....-26 For each k, he/she shifts all letters...

K=-1 ... M1 = WIQ

K=-2 ... M2 = VHP

Non Perfect Cipher Example

M = UGO (21 7 15)

K=+3

C = XJR (24 10 18)

Bruteforcing: k=-1-2-3.....-26 For each k, he/she shifts all letters...

K=-1 ... M1 = WIQ

K=-2 ... M2 = VHP

K=-3 ... M3 = UGO

“Toy” Perfect Cipher Example

M = UGO (21 7 15)

K=+5+2+2

C = ZIQ (22 9 18)

Bruteforcing...For each letter try all k=-1-1-1, -1-1-2,.....-26-26-26

“Toy” Perfect Cipher Example

M = UGO (21 7 15)

K=+1+2+3

C = VIR (22 9 18)

Bruteforcing...For each letter try all k=-1-1-1, -1-1-2,.....-26-26-26

M1=ADA

M2=XKR

M3=ELM

M4=UGO

Cryptanalysis: Breaking Ciphers

A real (non perfect) cryptosystem is **broken** if there is a way to break it that is **faster** than brute forcing.

Types of attacks:

- **Ciphertext attack**: analyst has only ciphertexts
- **Known plaintext attack:** analyst has a set of pairs plain-ciphertext
- **Chosen plaintext attack:** analyst can choose plaintexts and obtain their respective ciphertexts

Example: can you break this?

- I give you a ZIP-compressed file encrypted with a (secret) 4-bytes key
- I tell you how I encrypted it -- algorithm should not be secret (by Kerchoffs):
  - *C* = *K* xor *M*
  - Example:
    - K(hex) = **AA BB CC DD** (repeat the key)
    - M(hex) = `50 4B 03 04 BA DA 55 55 .. .. .. (and so on)`

      XOR
    - C(hex) = `FA F0 CF D9 10 61 99 88 .. .. .. .. .. .. ..`
- I give you a ZIP file encrypted with a key: can you recover the key w/o bruteforcing?

The Zip Example

***Algorithm**: C = K* xor *M*

- K(hex) = **AA BB CC DD** .. .. (repeat the key)
- M(hex) = 50 4B 03 04 BA DA 55 55 .. .. .. (and so on)

  XOR
- C(hex) = FA F0 CF D9 10 61 99 88 .. ..

PERFECT OR NOT ?

The Zip Example

***Algorithm**: C = K* xor *M*

- K(hex) = **AA BB CC DD** .. .. (repeat the key)
- M(hex) = 50 4B 03 04 BA DA 55 55 .. .. .. (and so on)

  XOR
- C(hex) = FA F0 CF D9 10 61 99 88 .. ..

NOT PERFECT -> len(k) < len(M) -> k is reused

The Zip Example:
Can you break this?

***Algorithm**: C = K* xor *M*

- K(hex) = **AA BB CC DD** .. .. (repeat the key)
- M(hex) = 50 4B 03 04 BA DA 55 55 .. .. .. (and so on)

  XOR
- C(hex) = FA F0 CF D9 10 61 99 88 .. ..

NOT PERFECT -> len(k) < len(M) -> k is reused

- *C = K* xor *M*
- *K = M xor C*
- *K = X X X X.... xor FA F0 CF D9....*

```
michele@starkiller ~/Desktop
➤ xxd test.zip | head
00000000: 504b 0304 1400 0808 0800 bb74 3150 0000  PK.........t1P..
00000010: 0000 0000 0000 0000 0000 1400 0000 6974  ..............it
00000020: 2d36 3931 3439 3678 3038 3931 3635 2e70  -691496x089165.p
00000030: 6466 ccbb 6554 5ccb ba36 8abb bbbb 5be3  df..eT\..6....[.
00000040: eeee ee6e 8dbb bb6b 20b8 8510 9ce0 ee2e  ...n...k .......
00000050: 0182 bb3b c125 b806 bb49 d65e 7baf 7dce  ...;.%...I.^{.}.
00000060: face ddf7 8cef c71d 73d4 acaa 77be 56d2  ........s...w.V.
00000070: b3bb 9ef1 34a5 b2b8 2423 0b13 3b1c e5b7  ....4...$#..;...
00000080: 9dc9 5938 0e12 6612 4753 1b12 387e 7e38  ..Y8..f.GS..8~~8
00000090: 80ba b713 9004 20e1 e526 a5e6 66e2 0684  ...... ..&..f...
```

```
michele@starkiller ~/Desktop
➤ xxd test2.zip | head
00000000: 504b 0304 1400 0808 0000 c051 3050 0000  PK.........Q0P..
00000010: 0000 0000 0000 0000 0000 3200 0000 5241  ..........2...RA
00000020: 4d53 4553 2046 494e 414c 2052 6576 6965  MSES FINAL Revie
00000030: 7720 5054 2050 6572 7370 6563 7469 7665  w PT Perspective
00000040: 204a 616e 2032 3020 2831 292e 7070 7478   Jan 20 (1).pptx
00000050: 504b 0304 1400 0600 0800 0000 2100 4fac  PK..........!.O.
00000060: d3c4 4302 0000 d416 0000 1300 0802 5b43  ..C...........[C
00000070: 6f6e 7465 6e74 5f54 7970 6573 5d2e 786d  ontent_Types].xm
00000080: 6c20 a204 0228 a000 0200 0000 0000 0000  l ...(..........
00000090: 0000 0000 0000 0000 0000 0000 0000 0000  ................
```

// MAGIC NUMBER", 1ST 4 BITES THAT DESCRIBES THE TYPE OF THE FIL

```
michele@starkiller ~/Desktop
➤ xxd test.zip | head
00000000: 504b 0304 1400 0808 0800 bb74 3150 0000  PK.........t1P..
00000010: 0000 0000 0000 0000 0000 1400 0000 6974  ..............it
00000020: 2d36 3931 3439 3678 3038 3931 3635 2e70  -691496x089165.p
00000030: 6466 ccbb 6554 5ccb ba36 8abb bbbb 5be3  df..eT\..6....[.
00000040: eeee ee6e 8dbb bb6b 20b8 8510 9ce0 ee2e  ...n...k .......
00000050: 0182 bb3b c125 b806 bb49 d65e 7baf 7dce  ...;.%...I.^{.}.
00000060: face ddf7 8cef c71d 73d4 acaa 77be 56d2  ........s...w.V.
00000070: b3bb 9ef1 34a5 b2b8 2423 0b13 3b1c e5b7  ....4...$#..;...
00000080: 9dc9 5938 0e12 6612 4753 1b12 387e 7e38  ..Y8..f.GS..8~~8
00000090: 80ba b713 9004 20e1 e526 a5e6 66e2 0684  ...... ..&..f...
```

```
michele@starkiller ~/Desktop
➤ xxd test2.zip | head
00000000: 504b 0304 1400 0808 0000 c051 3050 0000  PK.........Q0P..
00000010: 0000 0000 0000 0000 0000 3200 0000 5241  ..........2...RA
00000020: 4d53 4553 2046 494e 414c 2052 6576 6965  MSES FINAL Revie
00000030: 7720 5054 2050 6572 7370 6563 7469 7665  w PT Perspective
00000040: 204a 616e 2032 3020 2831 292e 7070 7478   Jan 20 (1).pptx
00000050: 504b 0304 1400 0600 0800 0000 2100 4fac  PK..........!.O.
00000060: d3c4 4302 0000 d416 0000 1300 0802 5b43  ..C...........[C
00000070: 6f6e 7465 6e74 5f54 7970 6573 5d2e 786d  ontent_Types].xm
00000080: 6c20 a204 0228 a000 0200 0000 0000 0000  l ...(..........
00000090: 0000 0000 0000 0000 0000 0000 0000 0000  ................
```

The Zip Example

NOW, WE HAVE THE CIPERTEXT, WE HAVE PART OF THE PLAINTEXT AND WE KNOW THE FUNCTION

Algorithm: *C = K* xor *M*

- K(hex) = AA BB CC DD .. .. (repeat the key)
- M(hex) = 50 4B 03 04 BA DA 55 55 .. .. (and so on)

  XOR
- C(hex) = FA F0 CF D9 10 61 99 88 .. ..

WE CAN EXTRACT THE KEY

- *K = M xor C*
- *K =* 50 4B 03 04 *xor* FA F0 CF D9 = AA BB CC DD ->
  - ATTACK?

The Zip Example

***Algorithm**: C = K* xor *M*

- K(hex) = AA BB CC DD .. .. (repeat the key)
- M(hex) = 50 4B 03 04 BA DA 55 55 .. .. (and so on)

  XOR
- C(hex) = FA F0 CF D9 10 61 99 88 .. ..

- *K = M xor C*
- *K =* 50 4B 03 04 *xor* FA F0 CF D9 = AA BB CC DD ->
  - KNOWN PLAINTEXT ATTACK

Computationally secure cryptography

IT IS A NON PERFECT CRYPTOGRAPHY BUT ITS BREAKNESS IS IMPRACTICAL IN TIME PERSPECTIVE

A more practical assumption

- Build a cipher so that a successful attack is also able to solve a hard computational problem efficiently
  - Solve a generic nonlinear Boolean simultaneous equation set
  - Factor large integers / find discrete logarithms
  - Decode a random code/find shortest lattice vector

Can we avoid assumptions?

- Open question: prove an exponential lower bound for the time taken to solve a hard problem, which has efficiently verifiable solution
  - it would shift from a computational security assumption to a theorem
  - ... and prove $P \neq NP$ as a corollary

DESIGN A CIPHERTEXT IN A MATHEMATICAL PROBLEM THAT IS EASY TO COMPUTE BUT HARD TO REVERSE HAVING OUTPUT ELEMENTS ONLY

Factor Large Integers (hints)

If p and q are two ***large primes***:

- computing $n = p * q$ is ***easy***
- but given n it is painfully ***slow*** to get p and q
  - quadratic sieve field, basically "try all primes until you get to the smaller between p and q"

Here "slow/difficult" means *"computationally very intensive"*, for all practical purposes the problem requires bruteforce over all possible values of x

Discrete logarithm (hints)

- If $y = a^x$ then $x = log_a y$ (Math 101)
- given x, a, p,
  - it is **easy** to compute $y = a^x \bmod p$,
  - but knowing y, it is **difficult** to compute x

Different problem than factorization, but it can be shown that they are related

Proving computational security

Outline of the method

1. Define the ideal attacker behaviour → UNDERSTAND HIS POSSIBLE CAPABILITIES AND KNOWLEDGE
2. Assume a given computational problem is hard
3. Prove that any non ideal attacker solves the hard problem → ABLE TO BREAK CIPHER BY SOLVING THE "HARD PROBLEM"

How to represent attacker and properties?

- Attacker represented as a program able to call given libraries → LIBRARIES THAT IMPLEMENTS THE CIPHER THAT YOU WANT TO TEST
- Libraries implement the cipher at hand → CIPHER OBSERVATION
- Define the security property as answering to a given question → CONFIDENTIALITY
- The attacker wins the game if it breaks the security property more often than what is possible through a random guess (SHANNON THEOREM)

Cryptographically Safe Pseudorandom Number Generators

PSEUDORANDOM FUNCTION: RANDOM GENERATOR BASED ON A DETERMINISTIC FUNCTION

Motivation and assumption

- We want to use a finite-length key and a Vernam cipher
  - We somehow need to "expand" the key
- We assume that the attacker can only perform poly(λ) computations

Definition

INPUT: FINITE LENGHT STRING $\downarrow$ OUTPUT: EXPANDED KEY

A CSPRNG is a deterministic function $\text{PRNG}: \{0,1\}^{\lambda} \to \{0,1\}^{\lambda+l}$ whose output cannot be distinguished from an uniform random sampling of $\{0,1\}^{\lambda+l}$ in $\mathcal{O}(\text{poly}(\lambda))$. l is the CSPRNG stretch.

HOW TO MODEL A CSPRNG

COMPLEXITY OF 2^K, WITH K GENERALLY HIGH

IF THE ATTACKER CAN BREAK THE KEY IN LESS THAN THAT TIME, THE ALGORITHM IS NOT SAFE

PROBLEMS: THERE EXIST A LOT OF PSRN THAT ARE EASY AND FAST TO BREAK, LIKE THE RANDOM FUNCTION IN C

CSPRNGs in practice

Existence

- In practice, we have only candidate CSPRNGs
  - We have no proof that a function PRNG exists
  - Proving that a CSPRNG exists implies directly P$\neq$NP

Practical constructions

- Building a CSPRNG "from scratch" is possible, but it is not the way they are commonly built (not efficient)
- Practically built with another building block: PseudoRandom Permutations (PRPs)
  - defined starting from PseudoRandom Functions (PRFs)

Random Functions (RFs)

! IT IS NOT A FUNCTION THAT GENERATES RANDOM NUMBERS. IT IS THE RESULT OF A RANDOM PROCEDURE WHICH CONTAINS PROPER INPUTS AND OUTPUTS

Randomly drawing a function

- Consider the set $\mathbf{F} = \{f : \{0,1\}^{in} \rightarrow \{0,1\}^{out}; in, out \in \mathbb{N}\}$
- A uniformly randomly sampled $f \xleftarrow{\$} \mathbf{F}$ can be encoded by a 2^{in} entries table, each entry out bit wide. $|\mathbf{F}| = (2^{out})^{2^{in}}$

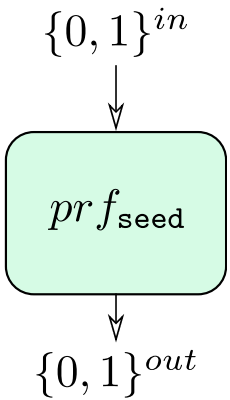
Toy example $in = 2, out = 1$

- $\mathbf{F} = \{f : \{0,1\}^2 \rightarrow \{0,1\}^1\}$ is the set of the 16 Boolean functions w/ two inputs
- Each one is represented by a 4-entry truth table
- Intuitively, the functions are 16 as there are $2^4 = 16 = (2^1)^{2^2}$ tables

Pseudorandom Functions (PRFs)

Definition

- A function $prf_{\text{seed}} : \{0,1\}^{in} \rightarrow \{0,1\}^{out}$ taking an input and a λ bits seed.
- The entire prf_{seed} is described by the value of the seed
- It cannot be told apart from a random $f \in \{f : \{0,1\}^{in} \rightarrow \{0,1\}^{out}\}$ in poly(λ)
- That is, if they give you $a \in \{f : \{0,1\}^{in} \rightarrow \{0,1\}^{out}\}$, you can't tell which one of the following is true
  - $a = prf_{\text{seed}}(\cdot)$ with $\text{seed} \xleftarrow{\$} \{0,1\}^{\lambda}$
  - $b \xleftarrow{\$} \mathbf{F}$, where $\mathbf{F} = \{f : \{0,1\}^{in} \rightarrow \{0,1\}^{out}\}$



Pseudorandom Permutations (PRPs)

Pseudorandom Permutation definition

- A bijective PRF: $prf_{\text{seed}} : \{0,1\}^{len} \rightarrow \{0,1\}^{len}$

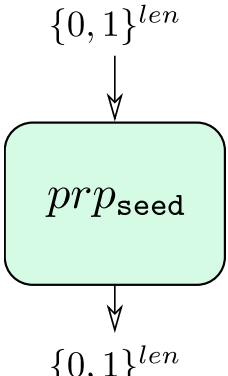
Wrapping your mind around it

- It is uniquely identified by the value of the seed
- It is not possible to tell apart in poly(λ) from a RF
- It's a permutation of all the possible $\{0,1\}^{len}$ strings

Operatively speaking

- acts on a block of bits outputs another one of the same size
- the output "looks unrelated" to the input
- its action is fully identified by the seed
  - Useful to think of the seed as a key

BIJECTIVE FUNCTION



IN AND OUT HAVE THE SAME LENGHT.
MATHEMATICAL PROPERTY THAT MAKES THE FUNCTION EASIER TO IMPLEMENT

Real world PRPs

The issue

- No formally proven PRP exists, yet
  - again, its existence would imply P$\neq$NP

Typical construction

1. Compute a small bijective Boolean function f of input and key
2. Compute f again between the previous output and the key
3. Repeat 2 until you're satisfied

In the real world...

Practical solution: public scrutiny

- Modern PRPs are the outcome of public contests
- Cryptanalytic techniques provide ways (=poly(λ) tests) to detect biases in their outputs: good designs are immune

PRPs a.k.a. Block ciphers

- Concrete PRPs go by the historical name of block ciphers
- Considered broken if, with less than 2^{λ} operations, they can be told apart from a PRP, e.g., via:
  - Deriving the input corresponding to an output without the key
  - Deriving the key identifying the PRP, or reducing the amount of plausible ones
  - Identifying non-uniformities in their outputs
- The key length λ is chosen to be large enough so that computing 2^{λ} guesses is not practically feasible

Keyspace and Brute Forcing

Keyspace generally measured in bits

- Attack time exponential on the number of bits (i.e., 33 bits need twice the time of 32)

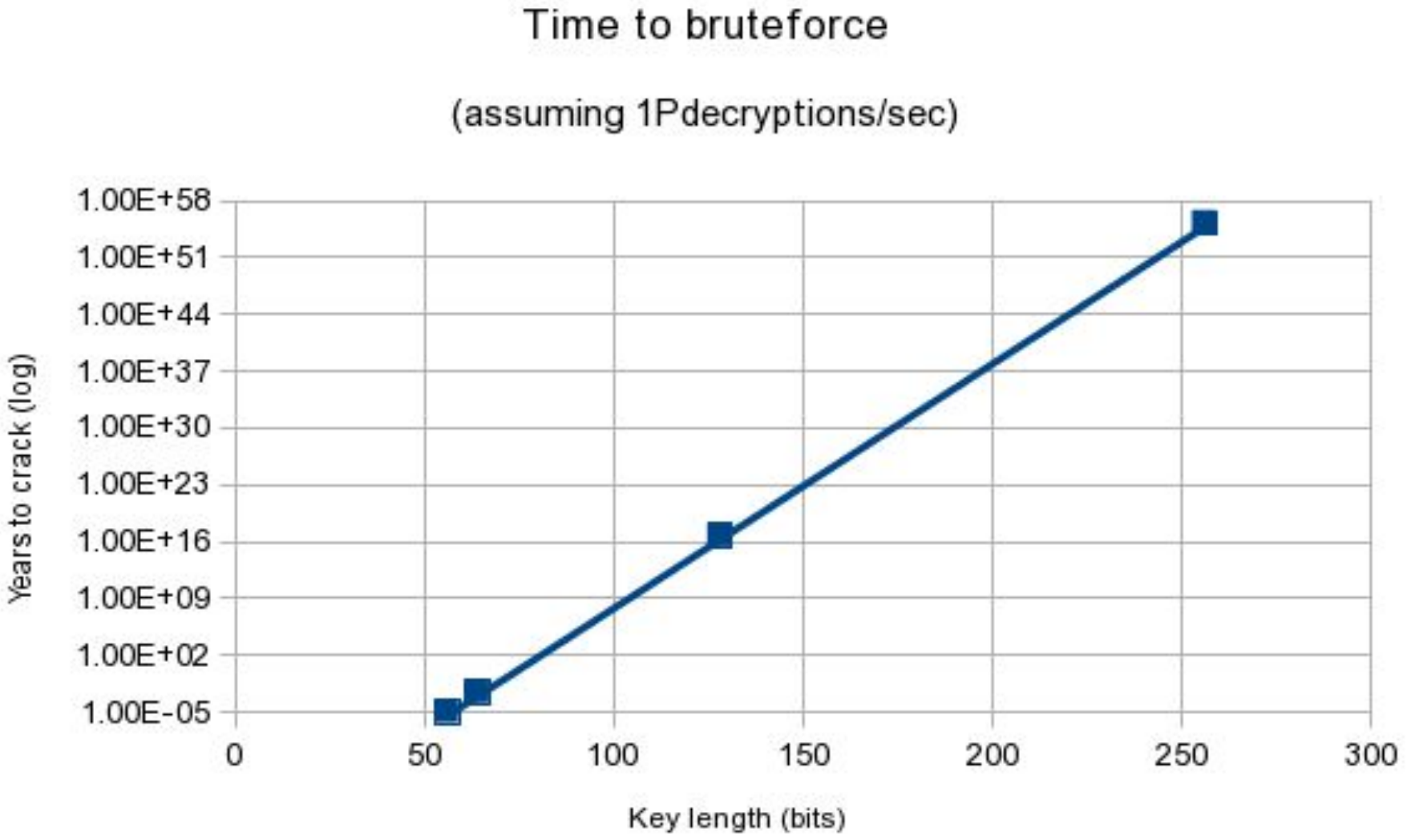
Basic Solution ?

Keyspace and Brute Forcing

Keyspace generally measured in bits

- Attack time exponential on the number of bits (i.e., 33 bits need twice the time of 32)
- Need to balance computational power vs key length.

Keyspace *vs.* Time for Brute Forcing



Quantifying computational unfeasibility

Boolean operations vs. energy to bring from 20 °C → 100 °C

- 2^{65} op.s ≈ an Olympic swimming pool
- 2^{80} op.s ≈ the annual rainfall on the Netherlands
- 2^{114} op.s ≈ all water on Earth

Practically acceptable unfeasibility

- Legacy level security: at least 2^{80} Boolean operations
- 5 to 10 years security: at least 2^{128} Boolean operations
- Long term security: at least 2^{256} Boolean operations

Widespread block ciphers

Advanced Encryption Standard (AES)

CURRENT STANDARD

- 128 bit block, three key lengths: 128, 192 and 256 bits
- Selected after a 3 years public contest in 2000-10-2 by NIST out of 15 candidates, re-standardized by ISO/IEC
- ARMv8 and AMD64 include dedicated instructions accelerating its computation (hitting 3+ GB/s)

IMPLEMENTATION OF A PSEUDORANDOM GENERATOR

Data Encryption Algorithm (DEA, a.k.a. DES)

PREVIOUS STANDARD

- Legacy standard by NIST (1977), the key is too short (56b)
- Patch via triple encryption, $\lambda = 112$ equivalent security → IN ORDER TO INCREMENT THE SECURITY MARGIN
- Still found in some legacy systems, officially deprecated

THE MARGIN WAS TO COMPUTE THE OPERATION 3 TIMES

NOWADAYS WE CAN FIND IT IN NON-UPDATED EMBEDDED DEVICES (SATELLITES...)

Case Study: DES

Originally designed by IBM (1973-1974)

Its core function is an S-box (a key-dependent substitution)

It uses a 56 bit key (2^{56} keyspace)

AND 64-BIT BLOCK

Case Study: DES vs. NSA

In 1976 it becomes a US standard; its S-boxes are "redesigned" by the NSA

- **Late 1980s:** *differential cryptanalysis* discovered
- **1993**: shown that the original S-boxes would have made DES vulnerable to the differential cryptanalysis, whereas the NSA-designed S-boxes were specifically immune to that.
- **Wait!** Wasn't differential cryptanalysis unknown until late 1980s? *Mmmmmmaybe* the NSA knew about differential crypto in the 70s.

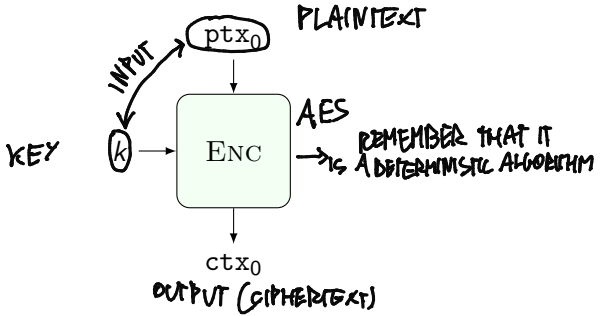
An Electronic CodeBook (ECB)

AES USES A SINGLE S-BOX THAT ACTS ON A BYTE INPUT TO GIVE A BYTE OUTPUT

A first attempt to encryption with PRPs

- It's ok to encrypt a plaintext $\leqslant$ block size with a block cipher

MAX OF 128b



PROBLEM

IF WE ENCRYPT SOMETHING LONGER THAN 128b AS PLAINTEXT?

SOLUTION

COMBINE A SEQUENCE OF BLOCKS...

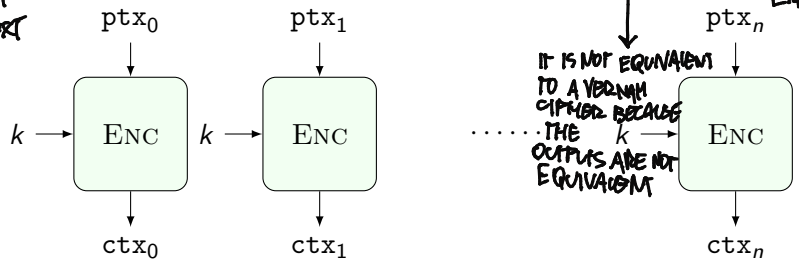
An Electronic CodeBook (ECB)

A first attempt to encryption with PRPs

- It's ok to encrypt a plaintext $\leqslant$ block size with a block cipher
- An extension to multiple blocks could be split-and-encrypt
  - Is it good (equivalent to Vernam fed with a CSPRNG)?

... AND SPLIT THE PLAINTEXT

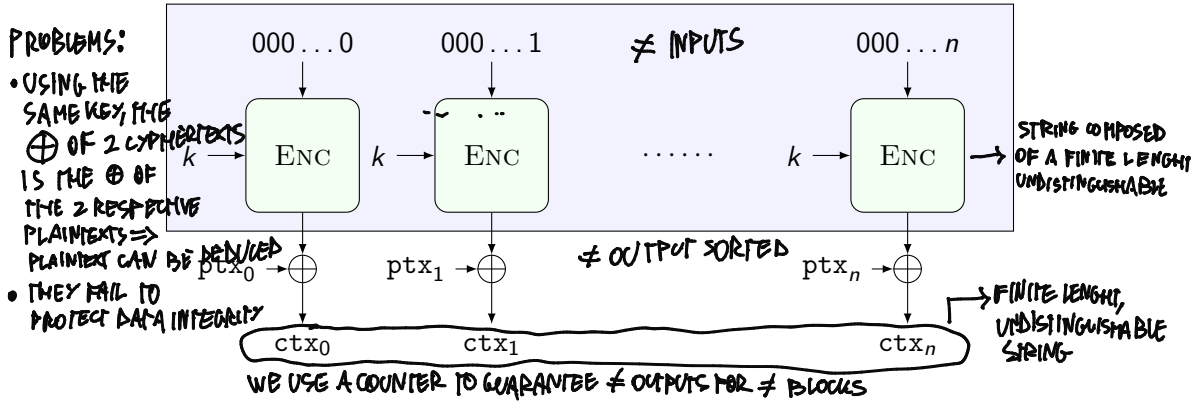
PROBLEM: WE ARE USING THE SAME KEY FOR EACH SINGLE BLOCK



Counter (CTR) mode

Getting it right

- The boxed construction is provably a PRNG if Enc is a PRP
- There is nothing special in the starting point of the counters



Raising the requirements

Confidentiality achieved ... for CoA

CIPHERTEXT ONLY ATTACKS

- Up to now, the attacker knew only ciphertext material

Confidentiality against Chosen Plaintext Attacks (CPAs)

- Our attacker knows a set of plaintexts which can be encrypted
- He wants to understand which one is being encrypted
- Ideal attacker: cannot tell which plaintext was encrypted out of two *he* chose (having the same length)
- Feels strange, but it happens with:
  - management data packets in network protocols (e.g., ICMP)
  - telling apart a encrypted commands to a remote host

SUPPOSE m_x AND m_y ARE DIFFERENT PLAINTEXTS (MESSAGE x, MESSAGE y). WE NEED THAT $Enc(m_x, m_z) = Enc$ BUT IT WILL NOT HAPPEN IN GENERAL

WHAT AN ATTACKER DO IS COMPARING THE SOLUTION OF $Enc(m_x, m_y) = \begin{cases} Enc_x \\ Enc_y \end{cases}$ WITH THE PREVIOUS RESULT TO UNDERSTAND WHICH MESSAGE IT IS

Achieving CPA Security

SOLUTION: MOVE FROM A DETERMINISTIC TO A NON DETERMINISTIC ALGORITHM

PROBLEM: WE WANT A NON DETERMINISTIC SOLUTION THAT CAN ALSO BE ENCRYPTABLE

No deterministic encryption

- The CTR mode of operation is insecure against CPA
  - The encryption is deterministic: same ptxs $\rightarrow$ same ctx

CHANGE THE KEY TO REACH THE BLOCK

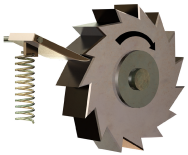
Decryptable nondeterministic encryption

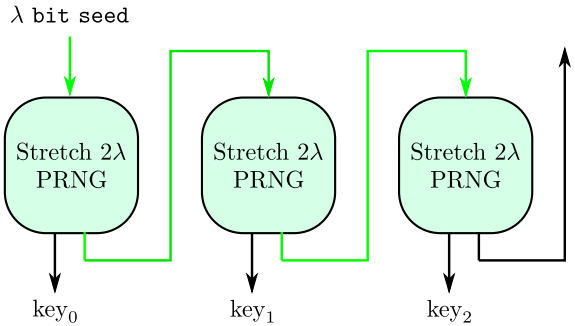
1. Rekeying: change the key for each block with a ratchet
2. Randomize the encryption: add (removable) randomness to the encryption (change mode of employing PRP)
3. Numbers used ONCE (NONCEs): in the CTR case, pick a NONCE as the counter starting point. NONCE is public

PROBLEM IN KEY MANAGEMENT

Getting it right

- The construction takes the name from the mechanical component: it is not possible to roll-back the procedure once you delete the value carried by green arrows

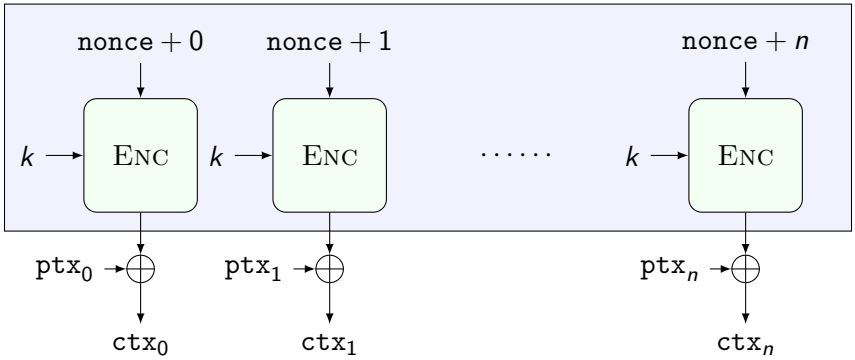




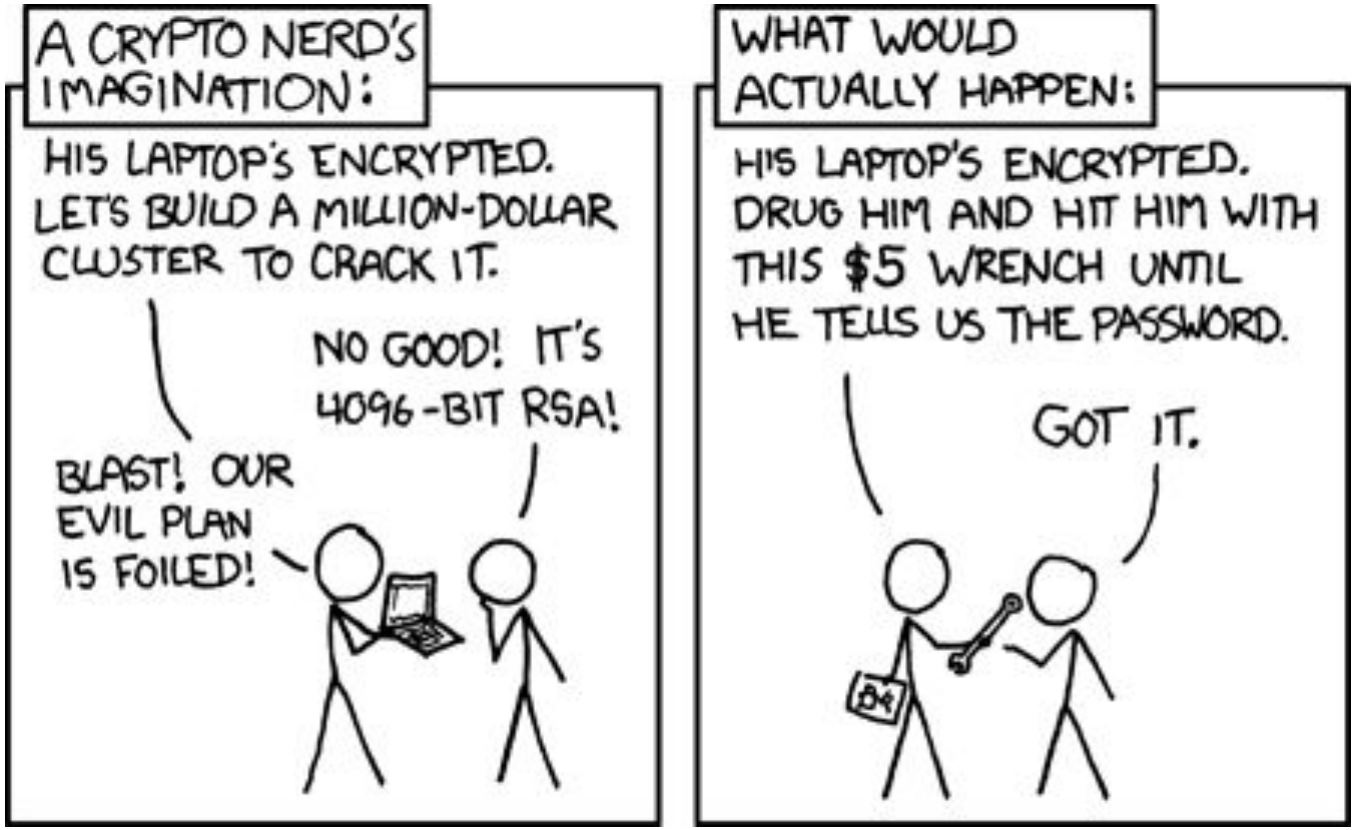
CPA-Secure Counter (CTR) mode

Getting it right

- Picking the counter start as a NONCE generates different bitstreams to be `xor`-ed with the `ptx` each time
- The same plaintext encrypted twice is turned into two different, random-looking, ciphertexts



A good point to remember...



Malleability and active attackers

ATTACKER THAT MAY TAMPER WITH THE CIPHERTEXT

WE KNOW WHAT HAPPENS TO THE PLAINTEXT

Malleability

- Making changes to the ciphertext (not knowing the key) maps to predictable changes in the plaintext
  - Think about AES-CTR and AES-ECB
- Can be creatively abused to build decryption attacks
- Can be turned into a feature (homomorphic encryption)

A CIPHERTEXT IS MALLEABLE IF, CHANGING THE CIPHERTEXT WE KNOW WHAT HAPPENS TO THE PLAINTEXT

EXAMPLE OF A MALLEABLE OPERATION: XOR $\oplus$

$P \oplus K = C \xrightarrow{\text{ATTACKER}} \tilde{C} = C + 1$

$\tilde{C} \oplus K = 1 + C \oplus K = 1 + P = \tilde{P}$

How to avoid malleability

- Design an intrinsically non malleable scheme (non trivial)
- Add a mechanism ensuring data integrity (against attackers)

Providing data integrity

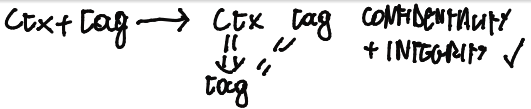
Confidentiality $\not\Rightarrow$ Integrity

- Up to now our encryption schemes provide confidentiality
- Changes in the ciphertext are undetected (at best)

Message Authentication Codes (MAC)

MECHANISM TO CHECK INTEGRITY

- Add a small piece of information (tag) allowing us to test for the message integrity of the encrypted message itself
  - Adding it to the plaintext and then encrypting is not good
- Nomenclature misleads: MACs do not provide data authentication



Definition

MAC USED JUST TO PROVIDE INTEGRITY

- A MAC is constituted by a pair of functions:
  - COMPUTE_TAG(string,key): returns the tag for the input string
  - VERIFY_TAG(string,tag,key): returns `true` or `false`

SHARED KEY

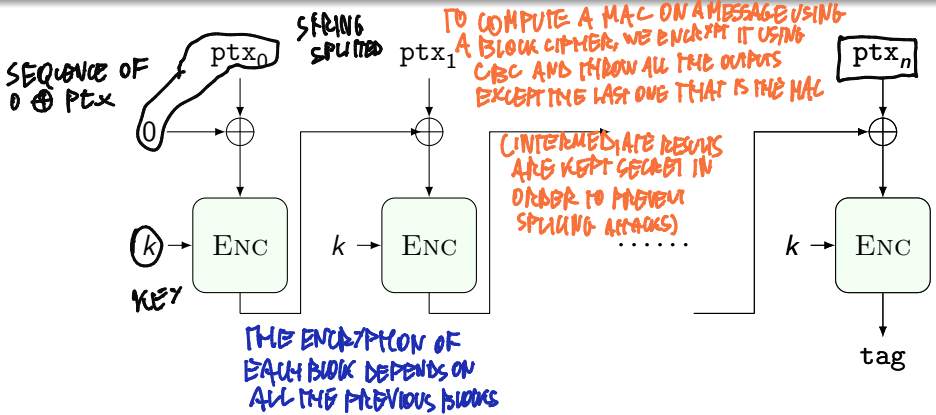
- Ideal attacker model:
  - knows as many message-tag pairs as he wants
  - cannot forge a valid tag for a message for which he does not know it already
  - forgery also includes tag splicing from valid messages
- N.B. the tag creating entity and the verifying entity must both know the same secret key
  - The tag verifier is able to create a valid tag too
  - ... and there goes the non-repudiation property

YOU ACTUALLY DON'T KNOW WHO SENT THE MESSAGE. STRING AND KEY ARE SHARED AND BOTH USERS CAN COMPUTE A TAG.

A THIRD USER CANNOT AUTHENTICATE THE OTHER

How to build a MAC? the CBC-MAC

CIPHER BLOCK CHAINING



Building a MAC with a PRP (block cipher)

- The CBC-MAC is secure for prefix free messages (why?) SET OF MESSAGES WITH THE SAME "INITIALS"
- Encrypting the tag once more fixes (provably) the issue

Practical MAC uses

Browser cookies

- HTTP cookies are a "note to self" for the HTTP server[a]
- The note should not be tampered between server reads
- Solution: server runs COMPUTE_TAG(cookie,k) and stores both the (cookie,tag)

[a] You can find a two slides cookies refresher at slides 40-41 of `https://polimi365-my.sharepoint.com/:b:/r/personal/10032133_polimi_it/Documents/FCI/2-Livello_Applicativo_v2020.pdf`

Later in the course

- Mitigating SYN-based denial of service attacks (SYN Cookies)
- Time-based two-factor authentication mechanisms (TOTP/HOTP)

Compressing for the sake of efficiency

Testing integrity

- Testing the integrity of a file requires us to compare it bit by bit with an intact copy or read it entirely to compute a MAC
- It would be fantastic to test only short, fixed length strings independently from the file size, representing the file itself
  - Major roadblock: there is a lower bound to the number of bits to encode a given content without information loss
- Can we build something close to the ideal scenario?

Cryptographic hashes

A pseudo-unique labeling function

- A cryptographic hash is a function $H : \{0,1\}^* \rightarrow \{0,1\}^l$ for which the following problems are computationally hard
  1. given $d = H(s)$ find s (1st preimage)
  2. given $s, d = H(s)$ find $r \neq s$ with $H(r) = d$ (2nd preimage)
  3. find $r, s; r \neq s$, with $H(s) = H(r)$ (collision)
- Ideal behaviour of a concrete cryptographic hash:
  1. finding 1st preimage takes $\mathcal{O}(2^d)$ hash computations guessing s
  2. finding 2nd preimage takes $\mathcal{O}(2^d)$ hash comp.s guessing r
  3. finding a collision takes $\approx \mathcal{O}(2^{\frac{d}{2}})$ hash computations
- The output bitstring of a hash is known as a digest

SU TERMINALE Gec-FileHash -a INDIRIZZO FILE

What is a Hash Function

IF THE FUNCTION IS KNOWN, AN ATTACKER CAN EASILY BREAK THE SYSTEM'S INTEGRITY

A function $H( )$ that maps arbitrary-length input x on fixed-length output, h

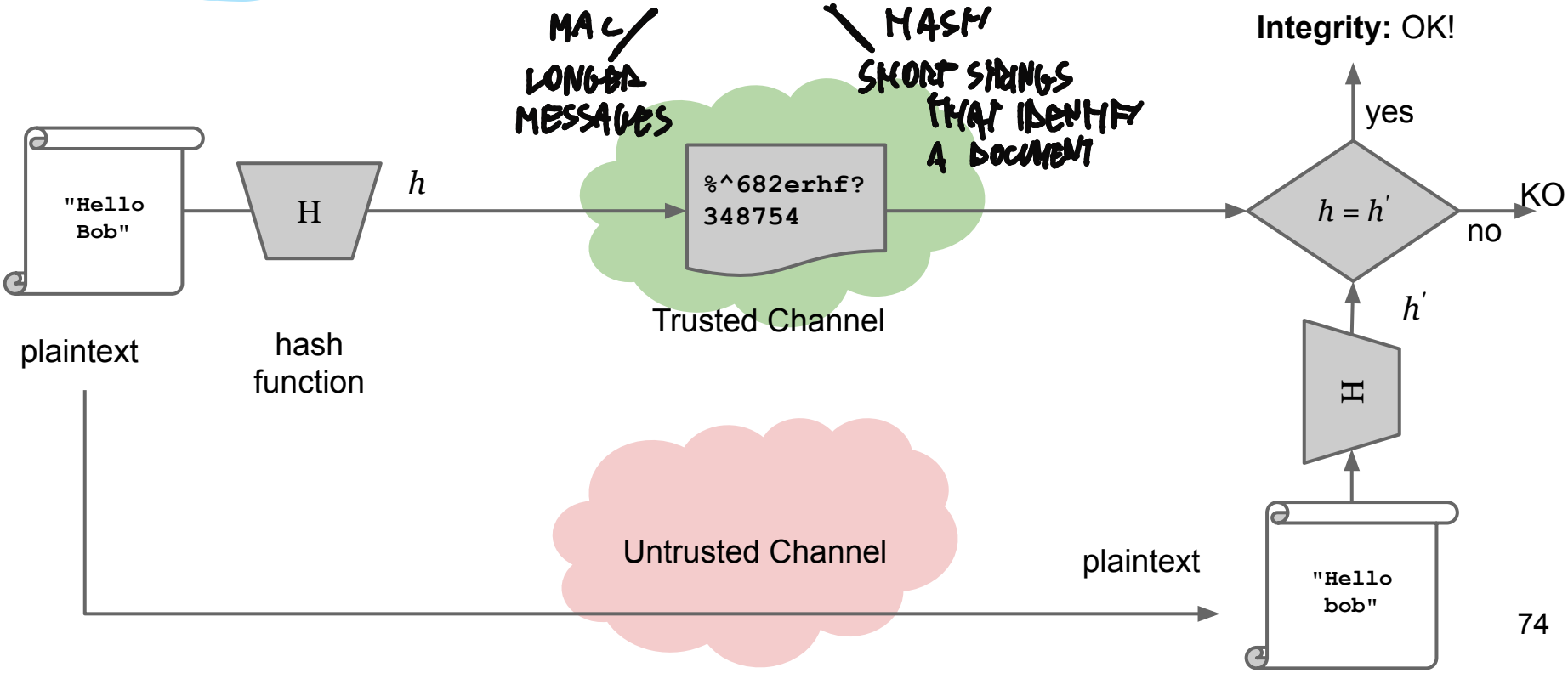
- Need to be **Fast**

ON THE OTHER HAND, MAC EXPLOIT A SECRET KEY UNDETECTABLE BY AN ATTACKER

TO SUM UP

MAC / LONGER MESSAGES

HASH / SHORT STRINGS THAT IDENTIFY A DOCUMENT



* THEY DEFINE A COMPUTATIONAL UPPER BOUND FOR PERFORMANCES THAT AN ATTACKER MUST SOLVE TO BREAK INTEGRITY

What is a Hash Function

A function $H( )$ that maps arbitrary-length input x on fixed-length output, h

- Need to be **Fast**
- **Collisions**: codomain "smaller" than domain. ⇒ ∃ 2 STRINGS THAT IDENTIFY THE SAME FILE

Computationally infeasible to find:

GIVEN h, IT MUST BE COMPUTATIONALLY HARD TO REVERSE THE FUNCTION

- input x such that $H(x) = h$ (a specific hash)
  - **preimage attack resistance**
- input y s.t. $y \neq x$ and $H(y) = H(x)$, with a given x (*)
  - **second preimage attack resistance**

FIXING AN INPUT, LOOK FOR ANOTHER ONE WITH THE SAME HASH

- couples of inputs $\{x, y\}$ s.t. $H(x) = H(y)$
  - **collision resistance**

LOOK FOR 2 INPUTS WITH THE SAME KEY

Attacks to Hash Functions (1)

Hash functions may be *broken*.

1. **Arbitrary collision or (1st or 2nd) preimage attack:**

Given a specific hash h, the attacker can find

x such that $H(x) = h$

or, equivalently, given a specific input x can find

y such that $y \neq x$ and $H(y) = H(x)$

faster than brute forcing.

With a n-sized hash function, *random* collisions can happen in (2^{n-1}) cases.

Attacks to Hash Functions (2)

2. Simplified collision attack:

The attacker can generate colliding couples

$$\{x, y\} \text{ s.t. } H(x) = H(y)$$

faster than brute forcing.

Random collisions can happen in $(2^{n/2})$ cases because of the birthday paradox:

- given *n* randomly chosen people, some pairs will have same birthday
  - probability = 100% if ***n* = 367**
  - probability = 99.9% if ***n* = 70** people
  - probability = 50% if *n* = 23 people, and so on...
- vs. very low chances that some of you are born on a specific date

"IT IS EASIER TO FIND 2 PEOPLE WITH THE SAME BIRTHDAY THAN CHOOSING ONE AND LOOK FOR SOMEONE WITH HIS SAME BIRTHDAY"

Concrete hash functions

What to use

- SHA-2 was privately designed (NSA), $d \in \{256, 384, 512\}$
- SHA-3 followed a public design contest (similar to AES), selected among ≈ 60 candidates, $d \in \{256, 384, 512\}$ → BASED ON HARDER MATHEMATICAL PROBLEMS; SLOWER
- Both currently unbroken and widely standardized (NIST, ISO)

What not to use

- SHA-1: $d = 160$, collision-broken [6] (obtainable in $\approx 2^{61}$ op.s)
- MD-5: horribly broken [7]. Collisions in 2^{11}, public tools online [5]
  - In particular, collisions with arbitrary input prefixes in $\approx 2^{40}$

d: 128 → 2^{128} → 2^{64}

FEW SECONDS USING A SIMPLE SMARTPHONE

Uses for hash functions

Pseudonymized match

- Store/compare hashes instead of values (e.g., Signal contact discovery)

MACs

- Building MACs: generate tag hashing together the message and a secret string, verify tag recomputing the same hash
- A field-proven way of combining message and secret is HMAC → HASH COMPUTED TWICE
  - Standardized (RFC 2104, NIST FIPS 198)
  - Uses a generic hash function as a plug-in, combination denoted as HMAC-hash_name
  - HMAC-SHA1 (!), HMAC-SHA2 and HMAC-SHA3 are ok ! IT MAY BE VULNERABLE IF BASED ON SHA-1 OR MD-5

Forensic use

- Write down only the hash of the disk image you obtained in official documents

Game changing ideas

Features we would like to have

- Agreeing on a short secret over a public channel
- Confidentially sending a message over a public authenticated channel without sharing a secret with the recipient
- Actual data authentication

Solution: asymmetric cryptosystems

- Before 1976: rely on human carriers / physical signatures
- DH key agreement (1976) / Public key encryption (1977)
- Digital signatures (1977)