CRYPTOGRAPHY: BASIC NOTIONS

Applied Cryptography

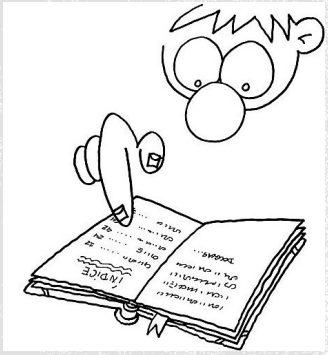
Silvio Ranise [silvio.ranise@unitn.it or *ranise@fbk.eu*]

UNIVERSITÀ DI TRENTO

FONDAZIONE BRUNO KESSLER

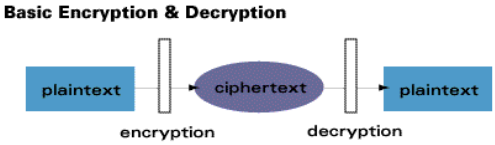
CONTENTS

- An informal overview
- Alice, Bob, and Eve
- Attacks
- Probability for cryptography
- Shannon's theorem
- Vernam cipher and beyond perfect ciphers



2 AN INFORMAL OVERVIEW

CRYPTOGRAPHY



- Used to provide
  - **confidentiality** and **integrity** of data
  - **authentication** and **anonymity** to communications
- By making information unintelligible to unauthorised parties
- Cryptanalysis = breaking encrypted data
- Cryptology = cryptography & cryptanalysis

A VERY PARTIAL AND RECENT HISTORY

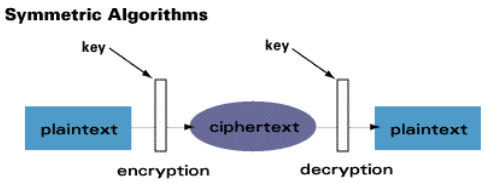
- During World War I, cryptology was dominated by military
- During early 1970s, cryptology was dominated by the government
  - computers were very expensive
  - government released very little information.
- Late 1970s/early 1980s, cryptology returned to academic/scientific communities
  - computers more readily available
  - demand for encryption increased due to fundamental changes in communication
- The increase in demand for cryptography was driven by industry
  - Financial services for secure electronic transactions
  - Businesses need to secure trade secrets stored on computers
  - Individual interest for e.g., secure wireless communications

BASIC PRINCIPLES

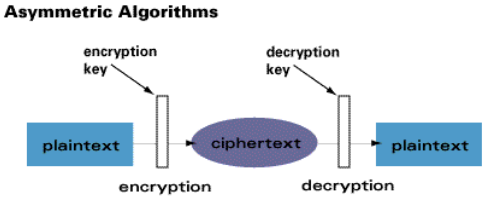
- **Security** should **not depend on the secrecy of the encryption algorithm, only the secrecy of the keys**
- Modern algorithms are based on mathematically difficult problems
  - E.g., prime number factorization, discrete logarithms, etc
  - There is no mathematical proof that these problems are in fact are hard, just empirical evidence
- The design of secure systems using encryption techniques focuses mainly on the **protection of (secret) keys**
  - Keys can be protected either physically or by encrypting them under other keys
  - Algorithm used to encrypt the data is made public and subjected to intense scrutiny

TYPES OF CRYPTOGRAPHY

- Symmetric = secret-key
  - Encrypt a single bit of plaintext at a time
  - Very fast



- Asymmetric = public-key
  - Take fixed number of bits (typically 64) and encrypt them as a single unit
  - Make a public key universally available, while only one possesses the private key
  - Data encrypted with public key, can only be decrypted with the private key (or viceversa)
  - It can serve to authenticate a source by using, e.g., a digital signature



AN APPLICATION OF CRYPTOGRAPHY

- Authentication & digital signatures
  - if Bob receives a message from Alice that has been encrypted with Alice private key and Bob is able to decrypt it using Alice public key, then Bob should feel reasonably certain that the message did in fact come from Alice
  - If Alice think it necessary to keep the message secret, she may encrypt the message with her private key and then with Bob public key, that way only Bob can read the message, and he will know that the message came from Alice
  - The only requirement is that public keys are associated their users in a trusted manner
    - Certificates infrastructure
- Pretty Good Privacy (PGP) is a software package that provides encryption and authentication for e-mail and file storage applications
  - See, e.g., https://www.openpgp.org/

WHAT CAN GO WRONG?

- Recall the basic principles
  - Security should not depend on the secrecy of the encryption algorithm, only the secrecy of the keys
  - Modern algorithms are based on mathematically difficult problems
- Example: consider RSA cryptography
  - Based on prime factorization
  - The size of the number being factored, called the modulus, determines how secure an actual use is
  - Intuitively, factoring large numbers takes more time than factoring smaller numbers, and the larger the modulus, the longer it would take an attacker to factor it
- However…

RSA FACTORING CHALLENGE (1991-2007)

- For each RSA number n, there exists prime numbers p and q such that $n = p \times q$
- Problem: find p and q primes, given only n
- Goal: measuring the **practical difficulty** of factoring large integers and cracking RSA keys used in cryptography
- *Moore law*: the **number of transistors** in a dense integrated circuit **doubles about every two years**

RSA number	Decimal digits	Binary digits	Cash prize offered	Factored on	Factored by
RSA-100	100	330	US$1,000[4]	April 1, 1991[5]	Arjen K. Lenstra
RSA-110	110	364	US$4,429[4]	April 14, 1992[5]	Arjen K. Lenstra and M.S. Manasse
RSA-120	120	397	US$5,898[4]	July 9, 1993[6]	T. Denny *et al.*
RSA-129 [**]	129	426	US$100	April 26, 1994[5]	Arjen K. Lenstra *et al.*
RSA-130	130	430	US$14,527[4]	April 10, 1996	Arjen K. Lenstra *et al.*
RSA-140	140	463	US$17,226	February 2, 1999	Herman te Riele *et al.*
RSA-150	150	496		April 16, 2004	Kazumaro Aoki *et al.*
RSA-155	155	512	US$9,383[4]	August 22, 1999	Herman te Riele *et al.*
RSA-160	160	530		April 1, 2003	Jens Franke *et al.*, University of Bonn
RSA-170 [*]	170	563		December 29, 2009	D. Bonenberger and M. Krone [***]
RSA-576	174	576	US$10,000	December 3, 2003	Jens Franke *et al.*, University of Bonn
RSA-180 [*]	180	596		May 8, 2010	S. A. Danilov and I. A. Popovyan, Moscow State University[7]
RSA-190 [*]	190	629		November 8, 2010	A. Timofeev and I. A. Popovyan
RSA-640	193	640	US$20,000	November 2, 2005	Jens Franke *et al.*, University of Bonn
RSA-200 [*] ?	200	663		May 9, 2005	Jens Franke *et al.*, University of Bonn
RSA-210 [*]	210	696		September 26, 2013[8]	Ryan Propper
RSA-704 [*]	212	704	US$30,000	July 2, 2012	Shi Bai, Emmanuel Thomé and Paul Zimmermann
RSA-220 [*]	220	729		May 13, 2016	S. Bai, P. Gaudry, A. Kruppa, E. Thomé and P. Zimmermann
RSA-230 [*]	230	762		August 15, 2018	Samuel S. Gross, Noblis, Inc.
RSA-232	232	768			
RSA-768 [*]	232	768	US$50,000	December 12, 2009	Thorsten Kleinjung *et al.*

10

REST OF THE COURSE…

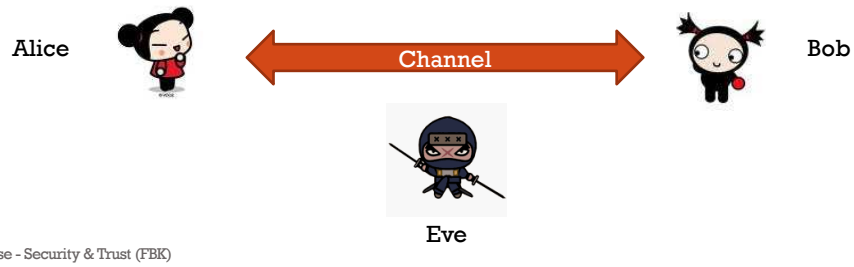
- Elaborate on these ideas…
- … understand how cryptographic algorithms are
  - Built
  - Broken
- … how they can be used in protocols

12

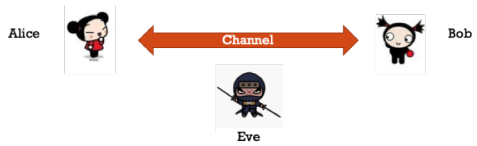
ALICE, BOB, EVE

IN A NUTSHELL

- Cryptography: krypto´s= "hidden" + graphia="writing"
- Goal of cryptography
  - Enable a secure communication between two users (**Alice** and **Bob**)
  - making it impossible for an eavesdropper (**Eve**) to understand the exchanged information



WHAT IS A CHANNEL?



- **Channel**
  - any physical or logical medium of communication from one user to another
- **Secure channel**
  - the information exchanged over it cannot be overheard or tampered with by eavesdroppers
  - A channel that is not secure is called insecure
- In **cryptography**, it is customary to assume that **channels are insecure**
- The **intent of cryptography** is to allow for a **secure communication on top of an insecure channel**

HOW CAN WE USE AN INSECURE CHANNEL TO TRANSFER MESSAGES SECURELY?

- Alice and Bob do not exchange their **plain** messages on the channel…
- … rather they transmit the messages in a **disguised** form
- Formally
  - **Plaintext** = the original message that Alice and Bob want to exchange
    - Notation: $\mathcal{P}$
  - **Ciphertext** = the disguised message transmitted over the channel
    - Notation: $\mathcal{C}$
- The idea is to **transform a plaintext into a ciphertext**, so that Alice sends the latter to Bob, and **Bob** is able to **reconstruct the plaintext** from the received ciphertext while this is very difficult (almost **impossible**) for **Eve**

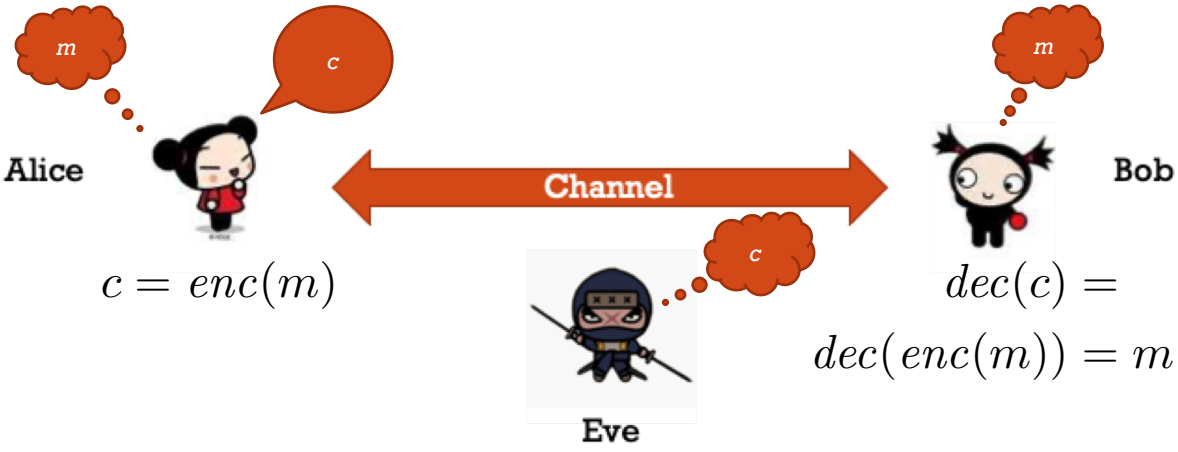
HOW TO REALIZE THE IDEA?

- Use a pair of functions
  - Encryption: $enc : \mathcal{P} \rightarrow \mathcal{C}$
  - Decryption: $dec : \mathcal{C} \rightarrow \mathcal{P}$

  Such that $dec(enc(m)) = m$ for every $m \in \mathcal{P}$

  - How Alice can send a message *m* to Bob without Eve to understand it?

- What is the relationship between *m* and *c?*
- Most importantly, is it possible to reconstruct *m* from *c?*



AN EXAMPLE

- $\mathcal{P} = \{0, 1, 2, ..., 9\} \quad \mathcal{C} = \{a, b, c, ..., i\}$

- $enc(0) = a \qquad dec(a) = 0$
  $enc(1) = b \qquad dec(b) = 1$
  $\ldots \qquad \ldots$
  $enc(9) = i \qquad dec(i) = 9$

- What is the relationship between *m* and *c?*
- Most importantly, **is it possible to reconstruct *m* from *c?***

REMARKS

- Alice and Bob agrees on the definitions of encryption and decryption without disclosing them to Eve
- Since Eve is only able to intercept ciphertexts, it should not be able to understand which plaintext Alice and Bob have exchanged…

- But, notice that
  - If the **sets of plaintexts and ciphertexts are too small**, then Eve can try all the plaintext-ciphertext pairs (**exhaustive search**)
  - Even if the sets of plaintexts and ciphertexts are large enough to make exhaustive search impractical, encryption and decryption can be defined in obvious way to allow Eve to easily reconstruct them (**guessing**)

- There is an additional and possibly more important problem…

MAIN PROBLEM

- The **definition of the encryption and decryption functions must be kept secret** so that Eve cannot know them
- In other words, they **must be exchanged on a secure channel**
  - All these difficulties make it **infeasible** for Alice and Bob to **agree** directly and secretly on the encryption and decryption functions, since they would spend too much in terms of performance of the channel

A (PARTIAL) WAY OUT

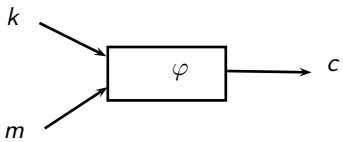
- New concept: **cryptographic key**
- We denote the set of (cryptographic) keys with $\mathcal{K}$
- We now consider a map

$$\varphi : \mathcal{P} \times \mathcal{K} \to \mathcal{C}$$

such that for every key $k \in \mathcal{K}$ the function

$$\varphi(\cdot, k) : \mathcal{P} \to \mathcal{C}$$

is an encryption function



A (PARTIAL) WAY OUT

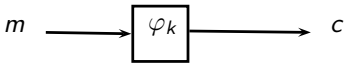
- New concept: **cryptographic key**
- We denote the set of (cryptographic) keys with $\mathcal{K}$
- We now consider a map

$$\varphi : \mathcal{P} \times \mathcal{K} \to \mathcal{C}$$

such that for every key $k \in \mathcal{K}$ the function

$$\varphi(\cdot, k) : \mathcal{P} \to \mathcal{C}$$

is an encryption function



Also written as

$$\varphi_k : \mathcal{P} \to \mathcal{C}$$

i.e. we consider a family of encryption functions (called a **cipher**) indexed over the set of keys

REMARKS

- Key differences between the two views
  - The definition of **φ** (**encryption algorithm**) can be very complex but it can be **public** (i.e. known to anyone) and Alice and Bob can agree on it over an insecure channel
  - The only component that must be kept **secret** (and thus exchanged over a secure channel) is the **cryptographic key *k***, that defines the encryption function to use
- A piece of information is secret (or private) if it must be communicated over a secure channel; otherwise it is public
- If we are also able to keep φ secret, then we obtain greater protection against eavesdroppers, provided that the encryption functions derived from the cipher are robust
- However…

KERCKHOFF'S PRINCIPLE

- *A cryptosystem should be secure even if everything about the system, except the key, is public knowledge*

- The **advantages** of exchanging only cryptographic keys rather than the cipher are as follows:
  - it is easier to keep secret k than φ
  - if the key is discovered, it is sufficient to change k and we do not need to renegotiate φ

- The main **disadvantage** is that the attacker can break the system just by finding k

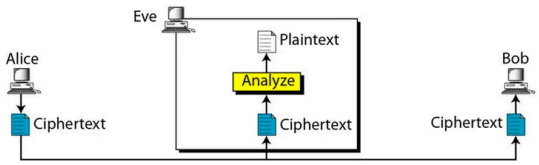
ATTACKS

ATTACKERS

- Depending on the **capabilities** of an attacker, we can distinguish among
  - CIPHERTEXT-ONLY
    - (also called KNOWN-CIPHERTEXT)
  - KNOWN-PLAINTEXT
  - CHOSEN-PLAINTEXT

Eve

CIPHERTEXT-ONLY (0)



The task of the attacker is very difficult and **a lot of computational power is required** to mount such an attack

- Attacker can only observe the ciphertexts
- Plaintexts are not available
- They know nothing about the plaintexts that generated the cyphertexts
- Totally passive, i.e. they can only intercept ciphertexts without modifying them

CIPHERTEXT-ONLY (1)

- Given
  - a finite set of ciphertexts $\{c_1, \ldots, c_N\} \subset \mathcal{C}$
  - a cryptographic key k in $\mathcal{K}$
  - a cryptographic function derived from a cipher φk
- The main goal of Eve is to **identify the corresponding set of plaintexts** $\{m_1, \ldots, m_N\} \subset \mathcal{P}$ where $c_i = \varphi_k(m_i)$

CIPHERTEXT-ONLY (2)

- Assume that Eve knows
  - The set of plaintexts
  - The set of ciphertexts
  - The finite set of ciphertexts $\{c_1, \ldots, c_N\} \subset \mathcal{C}$
- Assume that Eve knows neither the cipher nor the encryption function for the given key
- We can say that Eve's knowledge is too little to obtain any plaintext, but …
- … it could be enough to get some information about the plaintexts $\{m_1, \ldots, m_N\} \subset \mathcal{P}$
- **How?**

CIPHERTEXT-ONLY (3)

- For example, suppose that **two ciphertexts are equal** among those in $\{c_1, \ldots, c_N\} \subset \mathcal{C}$
- Then, Eve is able to conclude that also the corresponding plaintexts, from which they have been computed, are also equal

- If Eve has access to additional knowledge about the content of the messages, then it can derive more information
- In particular, **Eve** can obtain more information if it **knows** the probability **distribution of the plaintexts**

CIPHERTEXT-ONLY (4)

- Example
  - Let the set of plaintexts be the set of strictly positive natural number
  - Assume that 1 is transmitted very often, i.e. with a probability of 99% while the other plaintexts are transmitted very rarely
  - Let the set of ciphertexts contain the letters of the alphabets
  - Eve intercepts the following ciphertext: `xxxxaxxxAxxxxxxxxxxxcxxx`

- What can Eve deduce?

CIPHERTEXT-ONLY (4)

- Example
  - Let the set of plaintexts be the set of strictly positive natural number
  - Assume that 1 is transmitted very often, i.e. with a probability of 99% while the other plaintexts are transmitted very rarely
  - Let the set of ciphertexts contain the letters of the alphabets
  - Eve intercepts the following ciphertext: `xxxxaxxxAxxxxxxxxxxxcxxx`
- What can Eve deduce?
- Eve guesses that x, being the most probable cipher, corresponds to the plaintext 1, since 1 occurs more often than the others
- Thus it concludes that x is the plaintext that can be obtained by applying the encryption function to 1

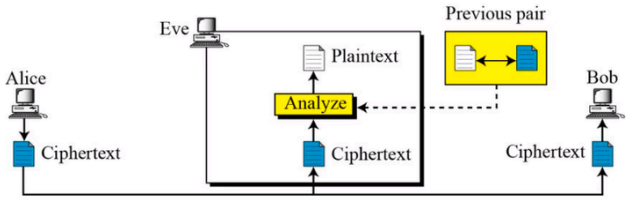
CIPHERTEXT-ONLY (5)

For a worked out example of this type of attack, see https://crypto.interactive-maths.com/frequency-analysis-breaking-the-code.html#

- Even if Eve does not know anything about the other plaintexts (like those corresponding to the ciphertexts a, A, c in the example), it is still able to recover 99% of the plaintext
- Summarizing
  - if an attacker knows the probability distribution of the plaintexts, it could obtain a lot of information merely by observing some ciphertexts
  - This holds under the assumption that all plaintexts are encrypted with the same cipher and the same key
  - The method may be difficult (if possible at all) to apply to short messages or messages that contain words with many occurrences of letters with low frequencies

KNOWN-PLAINTEXT (0)

This attack has a significantly better chance of success than CIPHERTEXT-ONLY attack



- Attackers know the plaintext that generates the ciphertext
- They cannot select the plaintext, but they can observe plaintext-ciphertext pairs
- We will see an example of this attack when considering ciphers based on the XOR operation later on

KNOWN-PLAINTEXT (1)

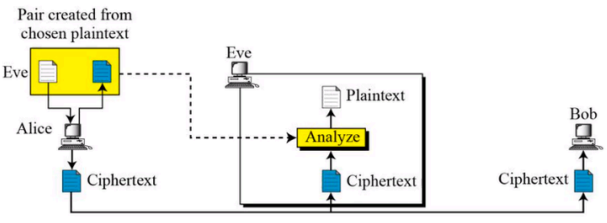
- Assume that Eve
  - has obtained a set of ciphertexts $\{c_1, \ldots, c_N\}$
  - with their corresponding plaintexts $(m_1, c_1) \ldots, (m_N, c_N)$
  - has full knowledge of the cipher
- The main goal of Eve is to **recover the cryptographic key**
  - I.e. going much beyond what is typically possible with ciphertext-only attacks since when Eve gets the key: it will be able to encrypt or decrypt any message by knowing exactly the same information shared by Alice and Bob
- This kind of attacks are also called **Key-Recovery attacks**

KNOWN-PLAINTEXT (2)

- The knowledge of plaintext-ciphertext pairs may not be sufficient to get the key in a reasonable time with Eve's available resources
- However, Eve does not need to recover the key in order to break the system…
- An alternative goal for Eve is to **discover a functionally equivalent algorithm for encryption and decryption**
  - Eve might be able to design a cryptographic algorithm that, even without knowing the key *k*, produces the same result as those of the cipher with the key *k*
- This kind of attacks are known as **Global Deduction/Reconstruction attacks**

CHOSEN-PLAINTEXT (0)

This attack has a good probability of success



- Attackers can both encrypt and decrypt
- They can select plaintext, encrypt it, observe the ciphertext and then reverse the entire process
- Note that the attacker is not necessarily trying to find the plaintext, but rather is trying to decipher the secret key used to encrypt the plaintext
  - **Question**: what is the impact of this attack with respect to the previous ones?

CHOSEN-PLAINTEXT (1)

- Assume that Eve
  - can obtain the ciphertexts corresponding to an arbitrary set of plaintexts $\{m_1, \ldots, m_N\}$ of its choice
- Same (alternative) goals of the known-plaintext attacks
  - Key recovery
  - Global Deduction
- Yet another (less ambitious) alternative goal is the following
  - Eve may try to guess previously unknown plaintext-ciphertext pairs
  - A similar goal can be stated also for known-plaintext attacks
- Remark: the careful choice of plaintext-ciphertext pairs can simplify the attack

FEASIBILITY OF ATTACKS: COMPARISON

- It is crucial for Eve to design attacks that are feasible
- In a ciphertext-only attack
  - The attacker only needs to be able to **intercept the traffic** from the sender to the receiver
- In a known-plaintext and a chosen-plaintext attack
  - The attacker needs to be able to **access to a larger portion of the infrastructure** used by communicating parties
- Attacker **skills**: Ciphertext-only < Known-plaintext < Chosen-plaintext
- Attacker **efforts**: Ciphertext-only > Known-plaintext > Chosen-plaintext

40

SHANNON'S THEOREM

Or the definition of perfect ciphers…

BROKEN AND PERFECT CIPHERS (1)

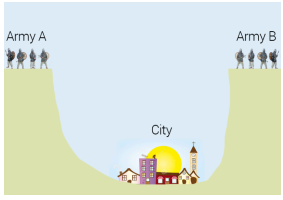
- A cipher is **broken** if a method of determining the plaintext from the ciphertext is found without being legitimately given the decryption key
- Any cryptosystem can be broken by an **exhaustive key search**
  - Known-plaintext and chosen-plaintext can be seen as refinement of this brute force method
- There exist ciphers that **cannot be broken** that are called **perfect**
  - Even exhaustive key search is of limited use for these ciphers
- An even more basic attack is **guessing plaintexts**, i.e. an attacker tries to guess a given plaintext
  - Indeed, the larger the space of possible plaintexts, the more difficult is to guess or, in other words, there is a very small probability to guess the right plaintext

BROKEN AND PERFECT CIPHERS (2)

- A cipher is **perfect** if, after seeing the ciphertext, an attacker gets no extra information about the plaintext other than what was known before the ciphertext was observed

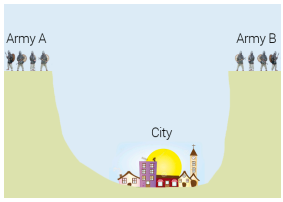
- Remarks
  - Attackers are not assumed to have *no information* about the plaintext
  - Indeed, they may know that the plaintext contain some kind of content (e.g., the PIN of a credit card)
  - The point is that the knowledge of the attacker about the plaintext is not increased after intercepting the ciphertext
  - In other words, the knowledge of the attacker about the plaintext is the same before and after the interception of the ciphertext

A SIMPLE EXAMPLE (1)



- Two armies (A and B) needs to coordinate an attack to a city or retreat
- The general of army A must decide and communicate the decision to army B
- The general of army A wants to protect the decision from eavesdroppers in the city
- Armies A and B randomly picks one encryption key among *K1* and *K2*
- After taking the decision, the general of army A sends
  - *E(K1,ATTACK) = 0* or *E(K1,RETREAT) = 1*
  - *E(K2,ATTACK) = 1* or *E(K2,RETREAT) =0*
- Upon reception of the message (either 0 or 1), army B retrieves the order by using the knowledge of the key previously selected

A SIMPLE EXAMPLE (2)



- Before intercepting the ciphertext, the eavesdropper has no idea of which decision will be taken
- After intercepting the ciphertext, the eavesdropper is faced with the following dilemma:
  - if the ciphertext = 0, then
    - if the key is K1, then the decision is ATTACK
    - if the key is K2, then the decision is RETREAT
  - if the ciphertext = 1, then
    - if the key is K1, then the decision is RETREAT
    - if the key is K2, then the decision is ATTACK
- **Regardless of which ciphertext was sent, the eavesdropper has not learnt anything useful about the plaintext since each plaintext remains equally likely**
- The cipher is **perfect but not secure** as eavesdropper has ½ probability to guess the decision!
  - It is desirable to increase the size of possible plaintexts to reduce the probability of guessing the right plaintext…
  - … but what does it mean in terms of keys that need to be used to cope with larger set of plaintexts?

45

MORE FORMALLY…

$\mathcal{K}$ is the set of keys

$\mathcal{P}$ is the set of plaintexts

$\mathcal{C}$ is the set of ciphertexts

φ is a cipher such that $\forall\, k \in \mathcal{K}$

$$\varphi_k : \mathcal{P} \to \mathcal{C} \text{ is an encryption function}$$

φ_k is invertible: there exists a unique φ_k^{-1} s.t. $\varphi_k^{-1}(\varphi_k(m)) = m$

SUMMARY OF NOTATION

if $m \in \mathcal{P}$, we use $P(m)$ to indicate the a priori probability that m occurs;

if $k \in \mathcal{K}$, $P(k)$ indicates the probability that k is the chosen key;

if $c \in \mathcal{C}$, $P(c)$ is the probability that c is the transmitted ciphertext;

$P(\mathcal{P})$ is the probability distribution of the plaintexts;

$P(\mathcal{C})$ is the probability distribution of the ciphertexts;

$P(\mathcal{K})$ is the probability distribution of the keys.

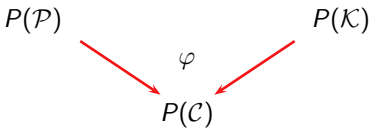
NEW NOTATION: PROBABILITY

PROBABILITY: BASIC ASSUMPTIONS

If there is a plaintext $\bar{m}$ such that $P(\bar{m}) = 0$, then it means that it never occurs and we can remove it from the set of the plaintexts. So, we can always assume that $P(m) > 0$ for every $m \in \mathcal{P}$.

In the same way, we assume that $P(k) > 0$ and $P(c) > 0$ for any $k \in \mathcal{K}$ and any $c \in \mathcal{C}$.

The two probability distributions $P(\mathcal{P})$ and $P(\mathcal{K})$ determine the probability distribution $P(\mathcal{C})$, because only one ciphertext can be obtained using the encryption algorithm φ with a given plaintext and key. Hence:



Definition

Let x and y be two events.

1. *The joint probability $P(x \wedge y)$ denotes the probability that x occurs* **and** *also y occurs.*
2. *The conditional probability $P(x \mid y)$ denotes the probability that x occurs* **given that** *y occurred.*
3. *We say that x and y are* **independent** *if $P(x \wedge y) = P(x)P(y)$.*

JOINT & CONDITIONAL PROBABILITY: DEFINITION

Joint probability and conditional probability are related by the formula:

$$P(m \mid c) = \frac{P(m \wedge c)}{P(c)}.$$

Interchanging m and c we have:

$$P(c \mid m) = \frac{P(m \wedge c)}{P(m)}.$$

which can also be reformulated as:

$$P(m \wedge c) = P(c \mid m)P(m) = P(m \mid c)P(c)$$

JOINT & CONDITIONAL PROBABILITY: RELATIONSHIP

INDEPENDENCE & CONDITIONAL PROBABILITY

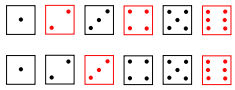
- The concept of **independence** formalises the perception that past events do not influence the outcome of future ones or provide any information about them
- In other words, if two events are independent, the order in which they occur is of no importance
- Yet in other words, independence of two events A and B means that the probability that B occurs is not changed if we "know" that A has occurred

- The notion of conditional probabilities formalise "the probability that B occurs if we know that A has occurred"
  - If two events are independent, then the probability that B (resp. A) happens if we know that A (resp. B) has happened is…

EXAMPLE (1)

- Assume we roll a fair dice, i.e. $P(X = 1) = P(X = 2) = \cdots = P(X = 6) = 1/6$
- Let us consider the following two events
  - x = an even number is rolled
  - y = a multiple of 3 is rolled
- It is easy to see that

$$P(x) = 1/2$$
$$P(y) = 1/3$$



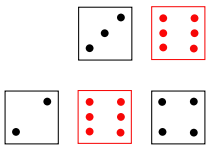
- Let us compute the joint probability that both x and y occur, i.e. that the number 6 is rolled

$$P(x \wedge y) = P(X = 6) = 1/6$$

EXAMPLE (2)

- Let us compute the following two conditional probabilities
  - x (an even number is rolled) occurs after y (a multiple of 3 is rolled) has occurred
  - y (a multiple of 3 is rolled) occurs after x (an even number is rolled) has occurred

$$P(x \mid y) = 1/2$$
$$P(y \mid x) = 1/3$$



- Observe that

$$P(x \wedge y) = P(x \mid y)P(y) = 1/2 \cdot 1/3 = 1/6$$
$$= P(y \mid x)P(x) = 1/3 \cdot 1/2 = 1/6$$

$$\forall\, m \in \mathcal{P},\ \forall\, k \in \mathcal{K}\ :\ P(m \wedge k) = P(m) \cdot P(k)$$

CLEVER BOY ASSUMPTION

- **Plaintext and keys are independent**

REMARKS ON CLEVER BOY ASSUMPTION

Alice and Bob choice of the cryptographic key is made independently from the messages that they intend to transmit

The assumption is realistic because the key is commonly chosen long before knowing which message will be sent

Indeed, if we already know the messages ... it is useless to encrypt them!

PERFECT CIPHER ACCORDING TO SHANNON

- Assume that **Eve can just intercept ciphertexts**
- We are **interested in the conditional probability that the plaintext *m* is sent, given that the ciphertext *c* is received**
- We **would like to have ciphers for which Eve cannot derive additional knowledge about a plaintext after intercepting a (single) given ciphertext** even knowing the probability distribution of the plaintexts

Definition

A cipher is called `perfect` *if*

$$\forall m \in \mathcal{P} \text{ and } \forall c \in \mathcal{C} \implies P(m) = P(m|c).$$

PERFECT CIPHER ACCORDING TO SHANNON

- Assume that **Eve can just intercept ciphertexts**
- We are **interested in the conditional probability that the plaintext *m* is sent, given that the ciphertext *c* is received**
- We **would like to have ciphers for which Eve cannot derive additional knowledge about a plaintext after intercepting a (single) given ciphertext** even knowing the probability distribution of the plaintexts

A perfect cipher is **unbreakable** if Eve can use only one ciphertext

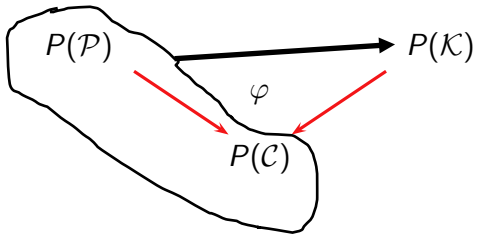
Definition

A cipher is called `perfect` *if*

$$\forall m \in \mathcal{P} \text{ and } \forall c \in \mathcal{C} \implies P(m) = P(m|c).$$

SHANNON THEOREM

$\mathcal{P} = \mathcal{K} = \mathcal{C} =$ set of n-bit strings



- Let us fix an integer *n*. Assume that
  - the keys have *n* bits,
  - the plaintexts have *n* bits
  - the ciphertext have *n* bits
- Assume also that
  - the plaintexts and the keys are independent (Clever boy assumption)
  - any *n*-bit string may be either a key or a plaintext
- φ is a perfect cipher if and only if both the following conditions hold
  1. the keys are perfectly random
  2. for any pair *(m,c)* of plaintext-ciphertexts, there is one and only one key *k* such that $c = \varphi(m,k)$

CONSEQUENCES OF SHANNON THEOREM

- If φ is a perfect cipher, then all ciphertexts have the same probability to be received
- Given any probability distribution of the plaintexts and even if such distribution is known, what Eve observes is a perfectly random cipher
- Hence Eve cannot recover any information on the sent message from the intercepted ciphertext
- **Warning**: this is true only if Eve intercepted **only one ciphertext**
- If Eve intercepts **more ciphertexts encrypted with the same key**, then the Shannon theorem no longer guarantees perfect secrecy

REMARKS

- A perfect cipher is thus not unbreakable in a practical sense
  - Because of the **strong assumption** that Eve can use only one ciphertext to break it
- One would like an **ideal cipher** to be **able to resist to any practical attack even using several ciphertexts**
  - We conclude that a perfect cipher is not necessarily ideal
- Any perfect cipher must have **a key space at least as large as its message space**
  - Example

    if Alice wants to send a 1GB file to Bob, then they must already share a 1GB key!

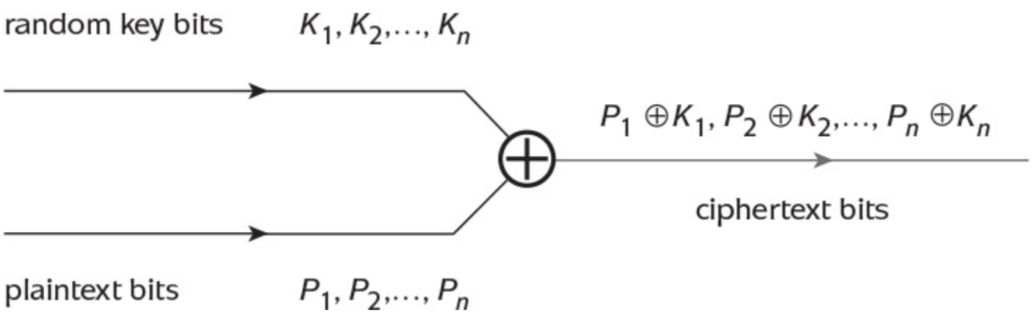
61

VERNAM CIPHER (ONE TIME PAD)

Cipher based on the eXclusive OR operation

x_1	x_2	x_1 XOR x_2
0	0	0
0	1	1
1	0	1
1	1	0

OVERVIEW OF VERNAM CIPHER



VERNAM CIPHER

What is the decryption function of the Vernam cipher?

- Also called **One-Time Pad**
- An interesting perfect cipher…
- We work under the same assumptions of the Shannon Theorem, namely that
  - the keys have n bits,
  - the plaintexts have n bits
  - the ciphertexts have n bits

  for some fixed value n

- In the Vernam cipher, we define encryption by means of the bitwise XOR (+) operation

$$c = \varphi_k(m) = k + m$$

i.e. c is the bitwise XOR of the binary strings k and m representing an arbitrary key and a plaintext, respectively

VERNAM IS PERFECT

- If the keys are perfectly random, then Vernam cipher is perfect

- To show this, observe that for any plaintext m and ciphertext c, **there exists a unique key *k*** such that $c = k + m$
- Namely, ***k* = *m*+*c*** since $m+c = m + k + m$ simplifies to $m + c = k + m + m$ that in turn simplifies to $m + c = k + 0 = k$ [Exercise: check this]
- By Shannon Theorem, we conclude that the Vernam cipher is perfect under the assumption that keys are randomly chosen

VERNAM IS PERFETCT BUT NOT IDEAL

There are 3 main disadvantages in using the Vernam cipher

1. the keys must be chosen perfectly at random (**randomness**)
2. the length of the key must be equal to the length of the message and the key must be exchanged secretly
   - such an operation is very expensive: **if we could guarantee the security of an exchanged key of the same length as the message, we may as well simply exchange the message itself*!***
3. each key must be used for one encryption only
   - If Eve knows a single plaintext-ciphertext pair $(m1,c1)$, the key can easily recovered by simply xor-ing $m1$ and $c1$ (recall that $c1 = m1 + k$ by definition of Vernam cipher)

$$m1 + c1 = m1 + (m1 + k) = (m1 + m1) + k = 0 + k = k$$

FROM PERFECT TO IDEAL

- **Requirement**: ability to encrypt a long message (e.g., a file of several megabytes) using a short key (e.g., a few hundred bits)
- Do not consider all possible adversaries, but only **computationally feasible adversaries**, that is, "real world" adversaries that must perform their calculations on real computers using a reasonable amount of time and memory
- This leads to a weaker definition of security called **semantic security**

- Since the focus is on the "practical," instead of the "mathematically possible," one shall also insist that the encryption and decryption functions are themselves **efficient algorithms** and **not just arbitrary functions**

67

SECURITY IN PRACTICE

SUMMARY ON PERFECT CIPHERS

- Since we assume that every ciphertext is generated with a different, randomly chosen, key, the only meaningful way to break perfect ciphers is to guess the plaintexts
  - So key recovery attacks do not matter here
- However, under the assumptions that keys are randomly chosen and used only once, plaintexts cannot be guessed as there is no additionally information that an attacker can get from intercepting ciphertexts
- This theoretically satisfying but in practice we will see that…

a theoretically secure cipher might not be secure in practice,

while a theoretically breakable cipher might be secure in practice

- This sounds puzzling but it is one of the root causes of the difficulty of applied cryptography*!*

A CRITIQUE OF ONE TIME PAD

- Key management issues
  - Key length → how to securely store large keys and how to securely **exchange** large **keys**
  - Key generation → how to randomly generate large keys
  - One time usage of keys → the above two expensive processes should be frequently repeated
- If secure exchange of keys is available, then why not to use it to exchange messages that have equal sizes of the keys
  - There are some cases in which key exchange is preferable… can you guess which one*?*
    - Hint… when it may be easier to distribute the key in advance by a convenient and secure means, potentially before the plaintext is even known. In the past it was the case of secret agents that after getting the key, typically exchanged in presence, they were able to decrypt a message when needed
- In practice, one time pad is unusable or useless and can only be used in high security environment and for very short messages…
  - As we will see, they are the basis for designing usable stream ciphers…

DESIDERATA FOR PRACTICALLY SECURE CIPHERS (1)

- Most of the ciphers we will see in this course will not be perfect and thus they are theoretically breakable
- However, they will satisfy a weaker and more practically relevant notion of security that is based on the following observation

most modern ciphers are regarded as secure in practice

because the known theoretical attacks take too much time to conduct

- In other words, to implement such theoretical attacks requires resources which are unrealistic for any attacker
- Characterizing the notion of practical security is not an easy task as it must consider several different aspects including…

DESIDERATA FOR PRACTICALLY SECURE CIPHERS (2)

- Characterizing the notion of practical security is not an easy task as it must consider several different aspects including…
  - **Cover time** → the time window in which a plaintext must be kept secret
    - This suggests that **no attack on the cipher can be conducted in less than the cover time**
    - This also implies that an **exhaustive key search takes longer than the cover time**
  - Notice that this aspect can consider only known, at the time of the analysis, attacks… if a new attack is discovered or other parameters are changed such as the available computation power, the evaluation should be repeated
  - **Computational complexity** → what computational processes are involved in known attacks on the cryptosystem how much time it takes to conduct these processes
    - For known ciphers, the computational processes are well understood since modern ciphers were adopted after a community effort using best practices such as adopting at least one process that is believed to be very computationally expensive
    - measuring the time taken to perform the processes requires a way of measuring the time it takes to run a process…
      - Do you know how to do this?

72

DIGRESSION ON TIME COMPLEXITY

TIME COMPLEXITY (1)

The time complexity measured in the number of comparisons is
$T(n)=n-1$

- What is the running time of the following algorithm to compute the maximum element in an array of integers?
- The answer depends on several factors
  - input
  - programming language and runtime environment
  - compiler, operating system, and hardware

```
max ← a[0]
for i = 1 to len(a)-1
    if a[i] > max
        max ← a[i]
return max
```

- It would be desirable to evaluate the **execution time** in a way that depends only on the **algorithm** and its **input** while abstracting away from the other factors
- This can be achieved by choosing an **elementary operation** which the algorithm performs repeatedly and define the **time complexity** $T(n)$
- In the case above, we want to identify the operations that the algorithm performs given an array of length n
  - It seems that the comparison a[i] > max is the elementary operation characterizing what the algorithm is repeatedly doing as the comparisons executed in the loop dominate all other operations (namely, assignment to max, incrementing the index variable i, and returning the final value of max). Additionally, the time to perform a comparison is constant as it does not depend on the size of the input array a

73

TIME COMPLEXITY (2)

- The **unit cost** is a simplified model for evaluating complexity where a number, of any size, fits within a memory cell and where standard arithmetic operations take constant time
- This is not the case if we consider bits and computations with larger numbers may take longer

- An **elementary operation** must satisfy the following two properties:
  - The algorithm cannot contain any other operations that are performed more frequently as the size of the input grows
  - The time to execute an elementary operation must be constant, i.e. it must be independent on the size of the input (as this grows), and is known as **unit cost**
- In some cases, it may be less clear which is the time complexity of an algorithm as elementary operations may or may not be executed depending on some condition in the program. For instance, consider the following algorithm to check if an element is in an array

```
for i = 0 to len(a)-1
    if x == a[i]
        return true
return false
```

- If x is not found in a, the algorithm makes n comparisons
- If x is equal to a[0], then there is only one comparison
- If x is equal to a[1], then there are just two comparisons
- …

TIME COMPLEXITY (3)

- There are typically two ways to determine the time complexity
- **Worst case behavior**
  - Let $T_1(n), T_2(n), \ldots$ be the execution times for all possible inputs of size n
  - The worst-case time complexity $W(n)$ is defined as $W(n) = \max(T_1(n), T_2(n), \ldots)$
  - For the algorithm searching for an element in an array, then we have that $W(n) = n$
  - This gives an *upper bound on time requirements*; the drawback is that it is often pessimistic
- **Average-case time complexity**
  - Let $T_1(n), T_2(n), \ldots$ be the execution times for all possible inputs of size n and $P_1(n), P_2(n), \ldots$ be the probabilities of these inputs
  - The average-case time complexity $A(n)$ is defined as $A(n) = P_1(n)T_1(n) + P_2(n)T_2(n) + \ldots$
  - Average-case time is harder to compute than worst case as it requires knowledge of how the input is distributed (sometimes it is difficult if possible at all to know the distribution of the input)

AN APPLICATION OF TIME COMPLEXITY (1)

- Consider the problem of reversing an array
- One would like to determine which one of the two algorithms on the right is better in terms of time complexity
- Algo 1
  - elementary operation: array assignment
  - since the outer loop is executed n-1 times, the worst case complexity is
    $W(n) = 1 + 2 + \ldots + (n - 1) = n(n - 1)/2 = n^2/2 - n/2$
- Algo 2
  - elementary operation: swap that is equivalent to 2 array assignments
  - since the loop is executed n/2 times, the worst case complexity is
    $W(n) = 2(n/2) = n$

Algo 1

```
for i = 1 to len(a)-1
    x ← a[i]
    for j = i downto 1
        a[j] ← a[j-1]
    a[0] ← x
```

Algo 2

```
for i = 0 to n/2
    swap a[i] and a[n-i-1]
```

```
aux = a[i]
a[i] = a[n-i-1]
a[n-i-1]=aux
```


AN APPLICATION OF TIME COMPLEXITY (2)

- Algo 1: $W(n) = 1 + 2 + \ldots + (n - 1) = n(n - 1)/2 = n^2/2 - n/2$
- Algo 2: $W(n) = 2(n/2) = n$
- Now, consider an array with 10,000 elements
  - Algo 1 will perform about 50,000,000 assignments
  - Algo 2 will perform instead only 10,000 assignments
  - So, Algo 2 is 5,000 times faster than Algo 1
  - Hence, we can say that Algo 2 should be our preferred implementation for reversing arrays
- For performing this kind of comparison in a more compact way, it is customary to use the so called Big O notation…

Algo 1

```
for i = 1 to len(a)-1
    x ← a[i]
    for j = i downto 1
        a[j] ← a[j-1]
    a[0] ← x
```

Algo 2

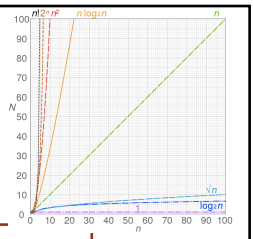
```
for i = 0 to n/2
    swap a[i] and a[n-i-1]
```

```
aux = a[i]
a[i] = a[n-i-1]
a[n-i-1]=aux
```

BIG O NOTATION (1)

- As seen above, the time complexity of algorithms can be expressed as a function of the size of the input that expresses the number of elementary operations performed
- The Big O notation is a convenient way to describe how fast a function, describing the time complexity of an algorithm, is growing
- *Definition*
  - Let T(n) and f(n) be two positive functions
  - we say that **T(n) has order of f(n)**, in symbols **T(n) ∈ O(f(n))**, if there are positive constants M and n_0 such that T(n) ≤ M*f(n) for all $n \geq n_0$
- *Intuition*
  - T(n) ∈ O(f(n)) means that T(n) does not grow faster than f(n)
- *Example*
  - Algo 1 has time complexity $O(n^2)$ whereas Algo 2 has time complexity $O(n)$ since
    - W(n) = $n^2/2 - n/2 \in O(n^2)$ for Algo 1
    - W(n) = $2(n/2) = n \in O(n)$ for Algo 2
  - It is thus immediate to see why Algo 2 is preferred over Algo 1

BIG O NOTATION (2)



- When T(n) ∈ $O(1)$, we say that the algorithm has **constant** time complexity
- When T(n) ∈ O(log n), we say that the algorithm has **logarithmic** time complexity
- When T(n) ∈ $O(n)$, we say that the algorithm has **linear** time complexity

Good to reasonable

- When T(n) ∈ O(n*log n), we say that the algorithm has **linearithmic** time complexity

Somehow still reasonable

- When T(n) ∈ $O(n^2)$, we say that the algorithm has **quadratic** time complexity
- When T(n) ∈ $O(n^k)$, we say that the algorithm has **polynomial** time complexity for k a constant greater than 2
- When T(n) ∈ $O(c^n)$, we say that the algorithm has **exponential** time complexity for c a constant greater than 2
- When T(n) ∈ $O(n!)$, we say that the algorithm has **factorial** time complexity

Less and less reasonable

REMARKS ON TIME COMPLEXITY

- A process can be conducted in *polynomial time* if the time taken to execute the process for an input of size *n* is not greater than *n^r*, for some number *r*
  - Informally, polynomial-time processes are 'quick' on all inputs of 'reasonable' size
- A process can be conducted in *exponential time* if the time taken to execute the process for an input of size *n* is approximately *a^n*, for some number *a*
  - Informally, exponential-time processes are 'slow' on all inputs of 'reasonable' size
  - As *n* increases, the length of time necessary to run the algorithm increases dramatically, until it is impossible in practice to compute the result
  - If a larger value of *a* is chosen, then this trend is exacerbated
- These are qualitative considerations as for instance, an exhaustive key search is quite feasible if *n* is small enough

81

END OF DIGRESSION ON EFFICIENCY

PRACTICAL ATTACK TIMES

- As already observed, time complexity provides an abstract notion of time which is based on computer operations
- To convert this into a real notion of time, we need to know how much real time it takes to run these basic computer operations
- In turn, this depends on the processor speed of the computer(s) used and other technological factors
- So, in practice we need to
  - estimate the computer speed (in terms of number of operations performed per second)
  - calculate the real time to conduct the process for an input of *n* bits by the formula
- Example
  - exhaustive key search has complexity *2^n*
  - assuming that we can perform one million operations per second
  - an exhaustive search for a 30-bit key will take $\frac{2\text{^}30}{10\text{^}6} = 1{,}000$ seconds

FINAL REMARKS

- Establishing the complexity of any known attacks is important and useful, but brings no guarantees of practical security
- This is so because of several reasons including
  - **undiscovered theoretical attacks**
  - **Complexity only deals with the general case**
  - **Implementation issues**
  - **Key management**
- Any real notion of practical security needs to consider also these issues
- In other words, the security of cryptographic applications needs to be evaluated in a broad context of use and the particular use case scenarios in which they are deployed

SUMMARY ON PERFECT CIPHERS (1)

- Assumptions
  - adversary has unbounded computational power (i.e. with infinite resources)
  - ciphertext onlyattacks, i.e. attacker can break the cipher using the cipher texts only
- Question: when is a cipher is perfect?
- Answer
  - The plain-text has a probability distribution
    - pP(x): A priori probability of a plain text
  - The key also has a probability distribution
    - pK(K): A priori probability of the key.
  - The cipher text y=eK(x) is generated by applying the encryption function eK
  - The plain texts and the keys have independent distributions

SUMMARY ON PERFECT CIPHERS (2)

- Attacker wants to compute a posteriori probability of plain text
- The probability distributions on P and K, induce a probability distribution on C, the cipher text.
  - For a key K, CK(x)={eK(x): x ∈ P}
- Does the cipher text leak information about the plain text?

  Given the cipher text y, we shall compute the a posteriori probability of the plain text, i.e. pP(x|y) and see whether it matches with that of the a priori probability of the plain text…
- A cipher has perfect secrecy if pP(x|y)=pP(x) for all x ∈ P, y ∈ C.
  - I.e., the a posteriori probability that the plaintext is x, given that the cipher text y is observed, is identical to the a priori probability that the plaintext is x

SUMMARY ON PERFECT CIPHERS (3)

- Suppose the 26 keys in the Caesar Cipher are used with equal probability 1/26
- Then for any plain text distribution, the Caesar Cipher has perfect secrecy
- Note that
  - P=K=C=Z26 and for $0 \leq K \leq 25$
  - Encryption function: $y=eK(x)=(x+k) \bmod 26$

- Perfect ciphers may not be very interesting from an application point of view… after all
- Better try to come up with ciphers where one key can be used to encrypt a large string of data and still provide some sort of security… in practice!