Computer Security and Privacy (COM-301)

Applied cryptography

Carmela Troncoso
SPRING Lab
carmela.troncoso@epfl.ch

Some slides/ideas adapted from: George Danezis, Yoshi Kohno

Important: you will not become a cryptographer

High level introduction to applied cryptography does not qualify you to design cryptographic primitives or protocols!

What you will learn?

What security properties different algorithms offer, and how can algorithms be combined to secure a system

What you will NOT learn?

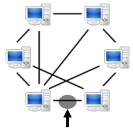
Cryptanalysis

How to prove formally that a scheme is secure

How to securely implement cryptographic schemes

To do these you need a cryptographer (or to become one)

Why does cryptography matter?



Data in transit

Data at rest

What would be the TCB?

The access control mechanisms we saw in the previous lectures require that there is a TCB that given the permissions (e.g., in the form of ACL, RBAC or Capabilities) and implements them. However, we **do not** always have a TCB. For instance while data is in transit, or stored in hardware that does not include a CPU, there is no TCB that can enforce access control.

Cryptography helps when there is not such a TCB. When using cryptography, the security of the system depends on the confidentiality and integrity of **cryptographic keys**.

What can we do with cryptography?

ENSURE SECURITY PROPERTIES

Cryptography can be used to ensure the **confidentiality** and **integrity** of data in transit or at rest

BUILD SECURE FUNCTIONALITY

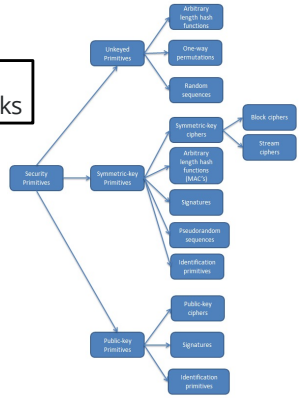
Cryptography can be used, among many others, to build **authentication** protocols**,** to protect from **denial of service,** or to support **anonymous** communications

Key Vocabulary

Cryptographic primitives
universal, exchangeable cryptographic building blocks

Secure functions where
- either you can't break it down any further or
- either there is no security argument for its individual parts

(What exactly a primitive is depends on the level of abstraction)

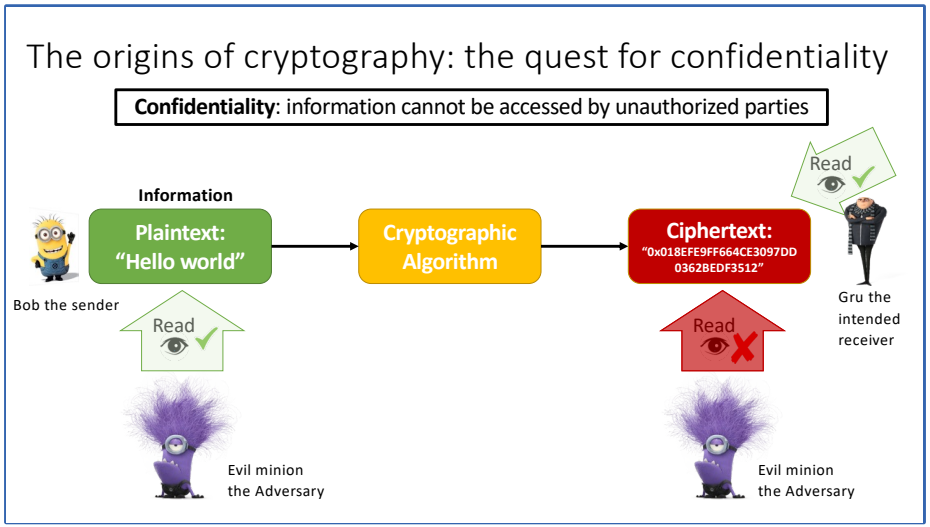


https://crypto.stackexchange.com/questions/39735/whats-a-cryptographic-primitive-really

https://i.stack.imgur.com/2yBJf.png

5

We call the "minimal algorithm" that implements a function a **cryptographic primitive**.
Minimal means that it cannot be further broken into pieces that carry out a crypto function, or that if you divide them into smaller sets of instructions then you can no longer build a security argument.



Key terminology:

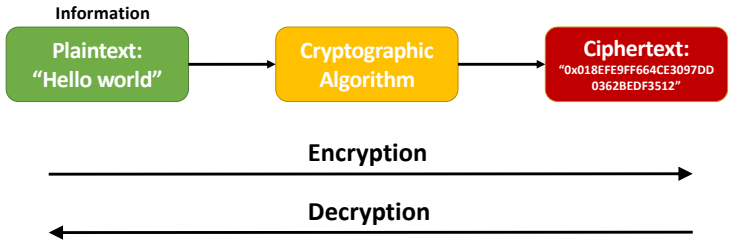
Plaintext: The original message written by the **sender** without encryption. It is readable by anyone who has access to it.

Cryptographic Algorithm: Set of mathematical instructions that encrypts and decrypts data

Ciphertext: Encrypted message obtained by applying the cryptographic algorithm to the **plaintext.** The intended receiver needs to first decrypt the message to be able to read it (next slide).

The origins of cryptography: the quest for confidentiality

Confidentiality: information cannot be accessed by unauthorized parties



As opposed to encoding, encryption cannot be reversed without a KEY

Encryption algorithm: Cryptographic instructions that convert a plaintext message into ciphertext

Decryption algorithm: cryptographic instructions that convert a ciphertext message into plaintext

Both encryption and decryption algorithm require a **key**

Cryptographic algorithms for confidentiality

1. Generate key **k** (and make sure intended receiver has it)
   Requires secure generation and sharing protocols

2. Encrypt message **m -> Enc(k,m)**

3. Send encrypted message **Enc(k,m)**

4. Decrypt message **Dec(k,m) -> m**

11

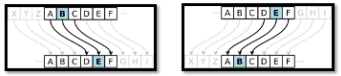
Cryptographic algorithms all follow a common schema with four steps
1) During the key generation & sharing phase, a key k is generated. The key must then be made available to both the sender and the **intended receiver.**
2) The sender has a message m (the plaintext) and encrypts it using the encryption algorithm. The key k is required to encrypt m
3) The sender sends the encrypted message (the ciphertext) to the intended receiver.
4) The intended receiver decrypts the ciphertext using the decryption algorithm and key k.

The first cryptographic algorithms

Caesar's cipher (50 BC)

Rotate the alphabet

Key: number of positions to shift (Julius Caesar used 3)



Encrypt Decrypt

hello world ⟶ khoor zruog

Kamasutra cipher (400 AD)

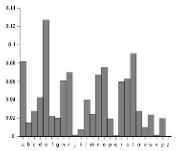
Permute the alphabet

Key: HOWBUGIACRYEVZXPJQMSNTFDKL

HOWBUGIACRYEV
ZXPJQMSNTFDKL

Encrypt/Decrypt: substitute by opposite letter

hello world ⟶ zkvvx pxfvy



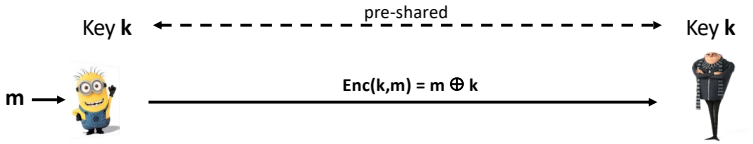
Problem??

Frequency analysis!

The problem with ciphers that transform letters deterministically into other letters is that the frequency of appearance of letters in the alphabet is preserved. Some letters appear more often than others, which reduce a lot which words could have generated the ciphertext.

Obtaining perfect secrecy: One Time Pad (OTP)

Key = string **k** of **random** bits **as long as the message**



Message	YEAH (ASCII Hex: 59454148)
Binary	01011001010001010100000101001000
OTP-Key ⊕	01110101000111010100101001001010
Encryption	00101100010110000000101100000010

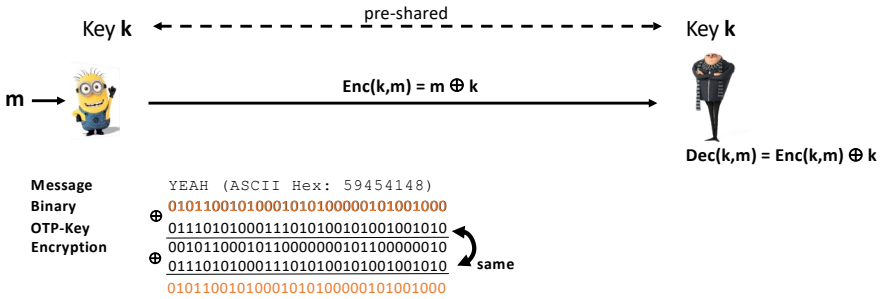
How to prevent frequency analysis? Make sure that letters are not encrypted to the same values.

One way of doing this is to have a **random** key as long as the message so that all parts of the message get encrypted to different values.
This long key is called a one-time-pad.

To encrypt a message one translates the message into bits, and then XOR these bits with the key.

Obtaining perfect secrecy: One Time Pad (OTP)

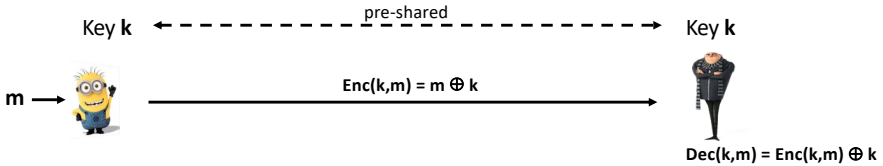
Key = string **k** of **random** bits **as long as the message**



To decrypt one XORs the encrypted message with the key (Recall: a⊕b⊕b=a)

Obtaining perfect secrecy: One Time Pad (OTP)

Key = string **k** of **random** bits as long as the message



Message	YEAH		YEAH
Binary (ASCII)	01011001010001010100000101001000	same	01001110010011110101000001000101
OTP-Key	01110101000111010100101001001010	different	11100001010000010111101011010001
Encryption	00101100010110000000101100000010		10101111000011100010101010010100

Delete "k" – it must never be reused!

$(msg1 \oplus pad) \oplus (msg2 \oplus pad) \rightarrow (msg1 \oplus msg2)$

- **Reveals where msg differ**
- **Frequency analysis works**
- **ASCII patterns (space or letter)**
  - **00-**
  - **01-**

One Time Pads are called like this because they should **not** be reused. If you reuse them, even though you cannot recover the full message, you can recover information about plaintexts. This information can be used to inform frequency analysis and help recover the messages themselves.

In particular, for ASCII characters one can use that spaces start with 00 and letters with 01 to detect whether a messages has a space:

2 letters = 01 ⊕ 01 = 00
2 spaces = 00 ⊕ 00 = 00
Letter + space = 01 ⊕ 00 = 01

Obtaining perfect secrecy: One Time Pad (OTP)

Why do we not use OTPs?

Key as long as the message (nowadays USBs contain several GB) and pre-shared! ← Moscow–Washington hotline

Key **must** be random!

"Each country delivered keying tapes used to encode its messages via its embassy abroad"

https://en.wikipedia.org/wiki/Moscow%E2%80%93Washington_hotline

Key **cannot** be reused

No integrity!

Problems:

- Keys as long as the messages themselves are very costly. What if you need to encrypt a high quality movie?
- Both sender and receiver need to know the full key. It is hard to secretly share such long strings of bits (also note that they cannot be transmitted in the same channel as the message, otherwise the adversary would have access to the key)
- The key has to be random (otherwise non-randomness can be used to gain information about the plaintext). It is **very** hard to generate long sequence of random numbers
- The key cannot be reused, so you need even more of these one time pads that are difficult to generate and share. Also, you need to make sure that they are destroyed so that they are never used twice.

- This mechanism only ensures confidentiality, but does not provide any integrity protection

Modern cryptography

Security should not depend on the secrecy of the encryption method (or algorithm), only the secrecy of the keys.

Modern algorithms are based on mathematically difficult problems - for example, prime number factorization, discrete logarithms, etc.

Modern cryptographic algorithms are too complex to be executed by humans.

Computer Security and Privacy (COM-301)

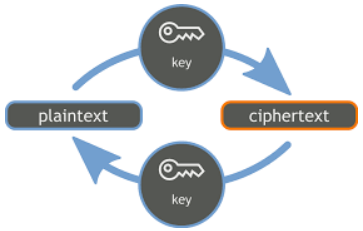
Applied cryptography
Symmetric encryption

Carmela Troncoso
SPRING Lab
carmela.troncoso@epfl.ch

Some slides/ideas adapted from: George Danezis, Yoshi Kohno

Symmetric encryption ciphers

Encryption of plaintext and decryption of ciphertext are done using
THE SAME KEY



Two types of ciphers:
- Stream ciphers
- Block ciphers

Integrity mechanism:
- Message Authentication Code (MAC)

28

First kind of encryption: Symmetric → both sender and receiver use **the same key**

There are two types of *Symmetric Encryption* Ciphers: stream ciphers and block ciphers.

We will also see one way of protecting the *integrity* of messages using symmetric encryption

What is a symmetric cryptographic key?

Fixed-size input to symmetric cryptographic primitives.

The size of the key influences the level of security provided

Key properties

Known to both parties

Partners must agree on the key **before** starting using the primitive

It is reused

The key is pre-shared once* and then reused

* keys do have a “duration”

It must be secret

Revealing the key eliminates any protection provided by the primitive

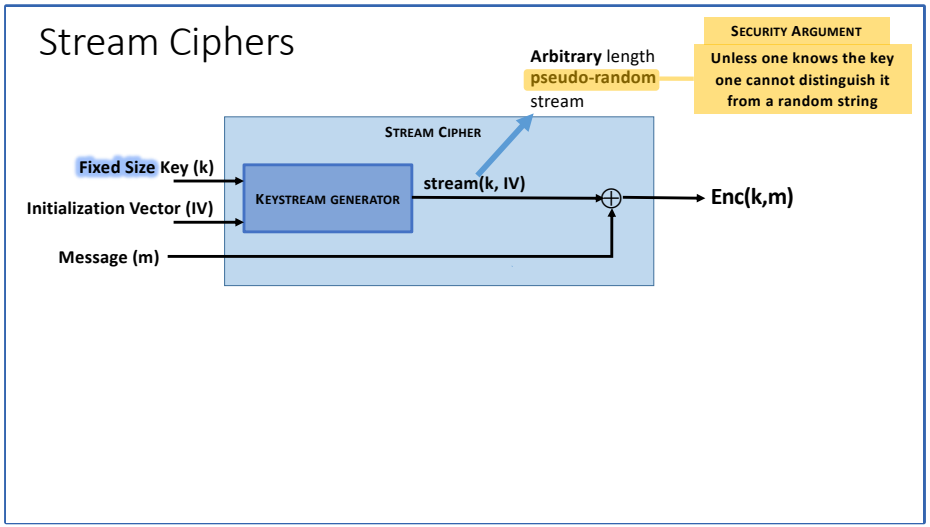
Computer Security and Privacy (COM-301)

Applied cryptography

Symmetric encryption - Confidentiality

Carmela Troncoso
SPRING Lab
carmela.troncoso@epfl.ch

Some slides/ideas adapted from: George Danezis, Yoshi Kohno



A stream cipher receives two inputs:

- A small (much smaller than the message!) key that must be kept secret
- An initialization vector that does not need to be secret (see next slide)

The keystream generator outputs a stream of bits that is pseudorandom (stream(k,IV), i.e., looks random for an adversary that does not have the key.

What is an Initialization Vector (IV)?

Initialization Vector: Fixed-size input to iterative cryptographic primitives

Important properties:

No IV reuse under the same key

Goal: messages encrypted with the same key look different (even the same message)

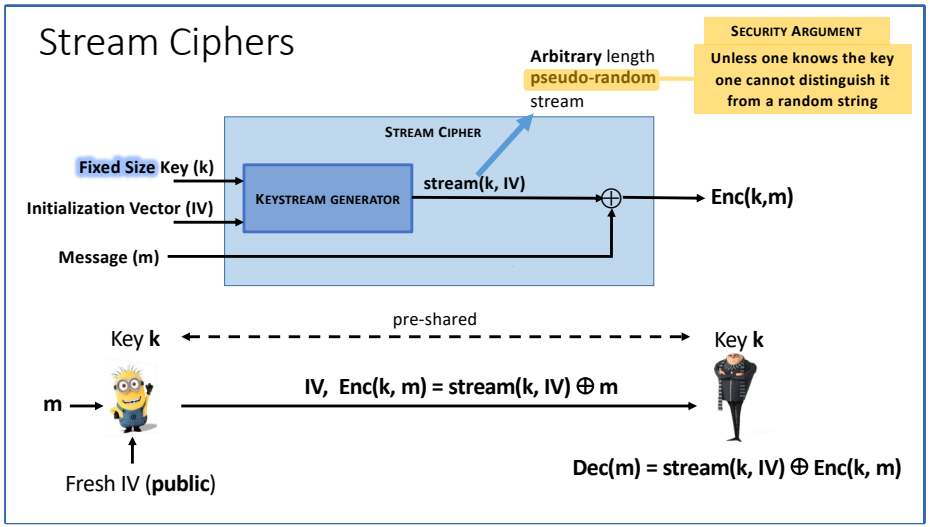
It **does not need to be secret**! Keeping the key secret is enough

But must be *unpredictable* in some block cipher modes

The IV **cannot** be reused. If the same IV is used more than once for a given key the keystream generator would produce the same stream of bits.
As stream-based encryption is based on XORing the message with the stream, if we use the same stream more than once we run into the same problems as when reusing a one time pad!

The IV **does not** need to be secret. The key is what protects the secrecy of the stream (without the key, only having the IV, one cannot recreate the stream)

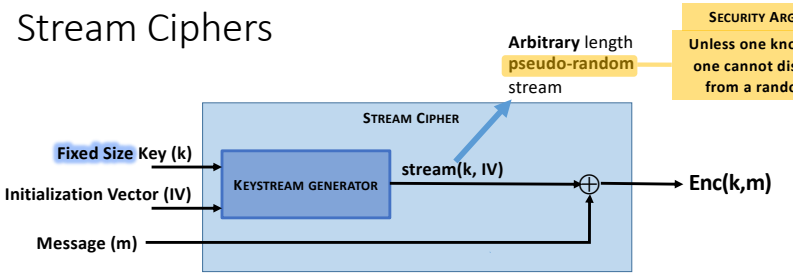
However, for some particular ciphers (in particular in some block cipher modes we will see below) it must be *unpredictable*. This means that given one IV, the adversary must not be able to guess how the next IV is going to look like. Otherwise, the adversary can use this knowledge to recover some plaintext seen in the past. See https://crypto.stackexchange.com/questions/3883/why-is-cbc-with-predictable-iv-considered-insecure-against-chosen-plaintext-atta for more details.



To **encrypt** the message, similar to the one time pad, we XOR the message bits with the output of the keystream generator.

To **decrypt**, the receiver repeats the operation: feeds the stream cipher with the shared key and the IV (notice it is sent with the encrypted message) to create the same stream. Then XORs this stream with the encrypted message to obtain the plaintext.

Stream Ciphers



Remaining downsides?

- ~~Key as long as the message (nowadays USBs contain several GB)~~ and pre-shared!
- ~~Key cannot be reused~~ Key **must** be random!
- No integrity

Better than before, though still necessary

Stream ciphers

STRENGTHS

Speed: algorithms are linear in time and constant in space

Low error propagation: errors in one bit do not affect subsequent symbols

WEAKNESSES

Low diffusion: all information of a plaintext symbol is contained in one encrypted symbol

Susceptibility to insertions/ modifications: text can be inserted, difficult to detect

Strengths:

Stream ciphers are very fast: the encryption time is linear with the size of the message.

If there is an error in encryption, because encryption is bit per bit, the error only affects one bit (contrast with block ciphers, see below)

Weaknesses: (contrast with block ciphers, see below)

As encryption is bit per bit, one can concentrate in small parts of the message to extract information about the corresponding plaintext

As encryption is bit per bit, it is hard to find out if anything has be modified, since the rest of the message is kept the same:

Imagine Gru knows the message is of the form:
"Pay XXXX to Bob"
Even when encrypted he can take the part corresponding to Bob and do

Stream(IV,K) xor (“Bob") xor (“Bob" xor “Gru") = Stream(IV,K) xor (“Bob" xor “Bob“) xor (“Gru") = Stream(IV,K) xor “Gru“!

Stream ciphers

STRENGTHS

Speed: algorith

Low error prop t symbols

WEAKNESSES

Low diffusion: a one encrypted symbol

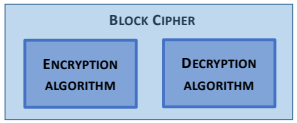
Susceptibility to fficult to detect

Don't design your own

Trivium **(80 bit key, < 4000 gates in HW)**
Salsa20 **(128/256 bit key, Random access)**

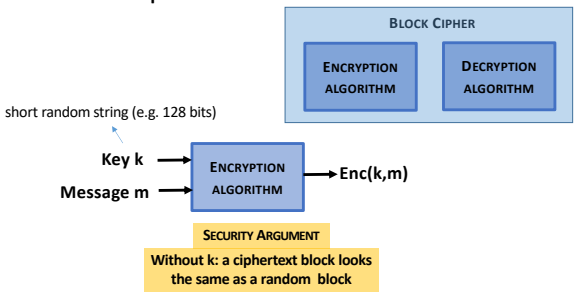
More stream ciphers: https://en.wikipedia.org/wiki/ESTREAM

Block Ciphers



Block ciphers are another symmetric cryptographic algorithm

Block Ciphers

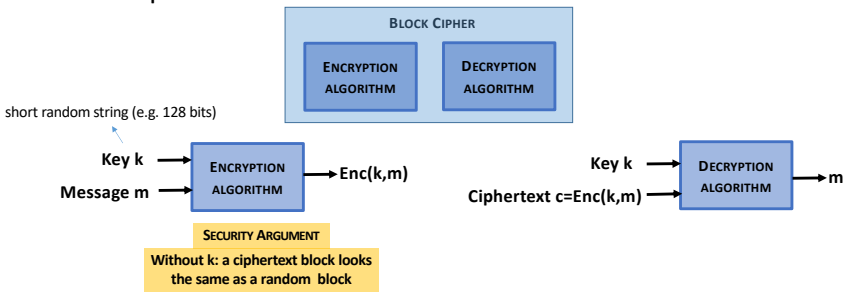


Encryption algorithm: Converts plaintext m to ciphertext c

The encryption algorithm of a block cipher **operates on small blocks**.

Encryption receives the key and a message block m and outputs an encrypted block of the **same size as the input.**
The encryption algorithm is such that given the output it looks random. In other words, the output is independent from the input.

Block Ciphers



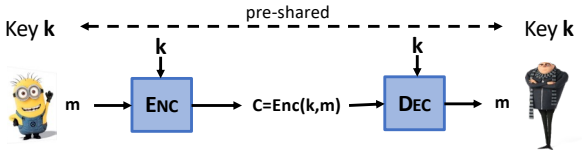
Encryption algorithm: Converts plaintext m to ciphertext c

Decryption algorithm: Concerts ciphertext c to plaintext m.

The inverse of Encryption → Dec(k; Enc(k; m)) = m

To **decrypt**, the receiver uses an algorithm Dec(), that is (generally) not the same as encryption. Given an encrypted block c and a key k, the decryption algorithm outputs the plaintext m.

Block Ciphers



The algorithms work on blocks that are the size of the key

Typically 128/256 bits

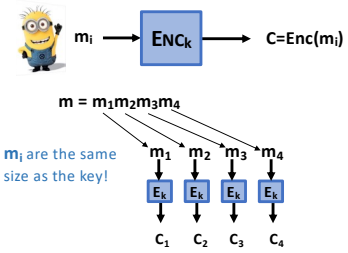
Messages are longer than a block! Requires iteration

Block ciphers' **mode of operation**

Messages are more than one block, so we need to have a way of chaining blocks together. This "chaining" is called a **mode of operation**.

Mode 1: Electronic Code Book (ECB)

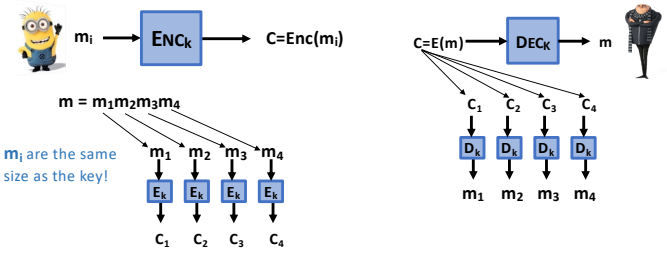
Straightforward scheme: encrypt & decrypt single blocks



Electronic Code Book (ECB) is a mode of operation in which blocks are encrypted *independently.*

Mode 1: Electronic Code Book (ECB)

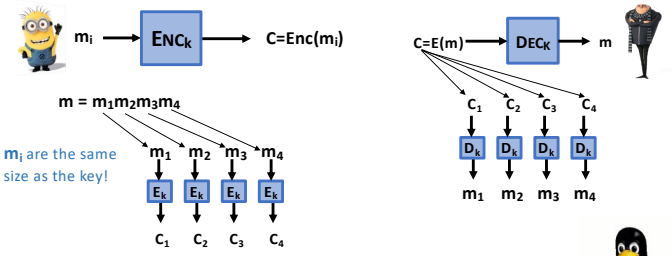
Straightforward scheme: encrypt & decrypt single blocks



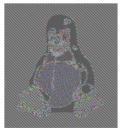
To decrypt, one also uses the decryption algorithm on individual ciphertext blocks.

Mode 1: Electronic Code Book (ECB)

Straightforward scheme: encrypt & decrypt single blocks



Problematic! $m_1=m_2 \rightarrow C_1=C_2$ **Don't use!!**

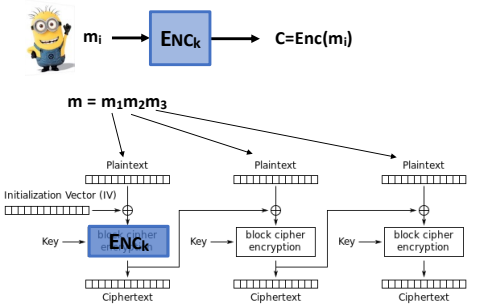


https://en.wikipedia.org/wiki/Block_cipher#/media/File:Tux_ecb.jpg

Thus, if two blocks in the message are the same, they will have the same appearance when encrypted!

Mode 2: Cipher Block Chaining (CBC)

Add IV and propagate information across blocks to introduce randomness



https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation#/media/File:CBC_encryption.svg

CBC Encryption

$C_0 = IV$

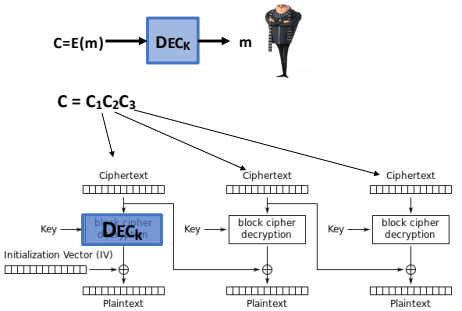
$C_i = Enc(k; m_i \oplus C_{i-1})$

Cipher Block Chaining is a mode of operation in which a block encryption depends on previous blocks (as defined in the formula and shown in the schema)

Note that every time a block is encrypted, by the properties of the block cipher, the ciphertext is random. Because the IV and/or the plaintext changes every time, this random value is different every time. Thus, when XORed with the next plaintext it creates a random string. If the IV changes, this is similar to a one-block one-time pad, but if one reuses the IV for the same plaintext and key, this value will be repeated.

Mode 2: Cipher Block Chaining (CBC)

Add IV and propagate information across blocks to introduce randomness



https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation#/media/File:CBC_encryption.svg

CBC Decryption

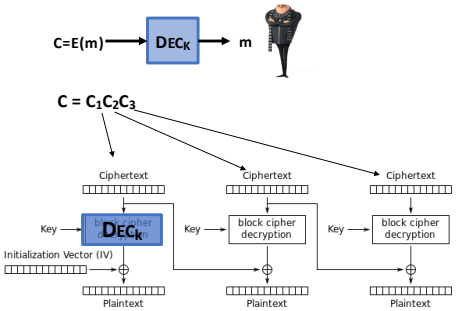
C_0 = IV

m_i = Dec(k; C_i) XOR C_{i-1}

To decrypt **Cipher Block Chaining** one has to do the inverse operation. Notice where the XOR with the IV and previous ciphertext happens as opposed to in the encryption algorithm.

Mode 2: Cipher Block Chaining (CBC)

Add IV and propagate information across blocks to introduce randomness



https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation#/media/File:CBC_encryption.svg

CBC Decryption

C_0 = IV

m_i = Dec(k; C_i) XOR C_{i-1}

What if IV is incorrect? The full decryption is wrong?

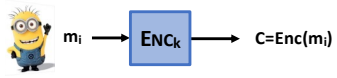
Can you decrypt a block alone? What do you need?

If IV is incorrect, only the first block is corrupted!! Note that the rest only depend on the ciphertext.

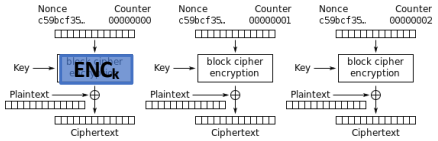
Blocks cannot be decrypted without having the ciphertext of the previous block! To recover a plaintext message block, the ciphertext of the previous block is XORed it with the output of the decryption on the current block.

Mode 3: Counter Mode (CTR)

Use increasing nonce to add randomness without dependencies between blocks



$m = m_1 m_2 m_3$



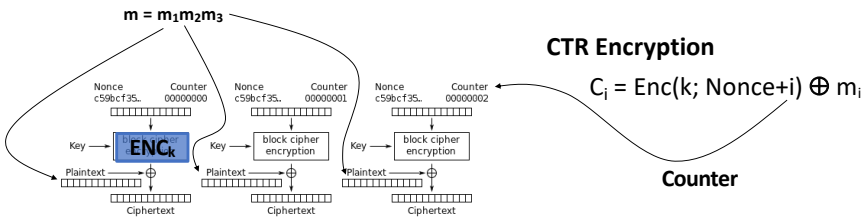
CTR Encryption

$C_i = Enc(k; Nonce+i) \oplus m_i$

In **Counter Mode**, we use a unique number composed by a **Nonce** and a **counter** so that the encryption receives a new number every time.
When encrypted, this number becomes a random one-time-pad that we XOR with the plaintext to obtain the ciphertext.

Mode 3: Counter Mode (CTR)

Use increasing nonce to add randomness without dependencies between blocks



CTR Encryption

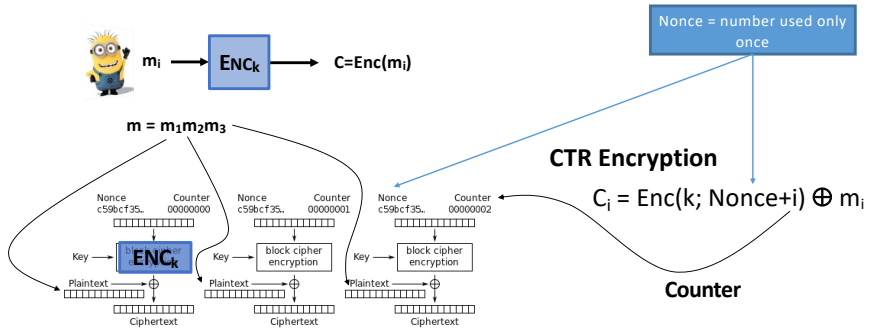
$C_i = Enc(k; Nonce+i) \oplus m_i$

Counter

The counter keeps increasing so that, if the nonce is new, the output of the block cipher for every block will be different.

Mode 3: Counter Mode (CTR)

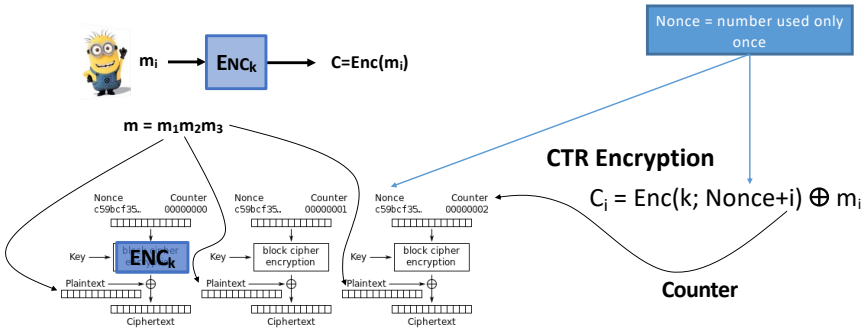
Use increasing nonce to add randomness without dependencies between blocks



Nonce = number used only once. It **must** be new every time. Otherwise we are feeding the encryption with the same number and we obtain the same random pad at the output: *effectively we would be reusing a one time pad!*

Mode 3: Counter Mode (CTR)

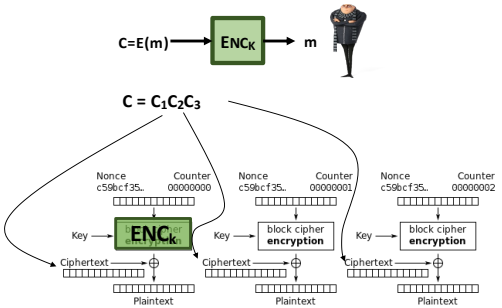
Use increasing nonce to add randomness without dependencies between blocks



Do we need the decryption algorithm??

Mode 3: Counter Mode (CTR)

Use increasing nonce to add randomness without dependencies between blocks



CTR Decryption

C_i = **Enc**(k; Nonce+i) ⊕ m_i

For counter mode we **do not need the Decryption algorithm.**

As CTR mode operates like a one time pad, to decrypt we need the same random sequence of bits, i.e., we need the same sequence which is obtained using the same algorithm: **encryption.**

Summary: Block ciphers

STRENGTHS

High diffusion: information from one plaintext symbol is diffused into several ciphertext symbols

Immunity to tampering: difficult to insert symbols without detection

WEAKNESSES

Slow: an entire block must be accumulated before encryption / decryption can begin

Error propagation: in some modes of operation errors affect several bits/blocks

*Different modes of operation offer different trade-offs and these weaknesses/strengths may actually not apply.

Strengths:

Because of the chaining and the entangling of blocks there is great diffusion: every bit affects several blocks

The chaining also helps to detect insertion or modification (integrity violations): if something changes it will be propagated and decryption will not make sense.

Weaknesses: (contrast with block ciphers, see below)

Block ciphers are slow compared to stream ciphers because they need to wait for a full block before encryption/decryption.

When there is an error, in some modes such as CBC where encryption is chained, the error gets propagated to other blocks

Summary: Modes of operation

Electronic Code Book (**ECB**)

- ✅ Directly encrypt and decrypt single blocks
- ❌ Large information leakage due to lack of randomness across ciphertext blocks

Cipher Block Chaining (**CBC**)

- ✅ Avoids ECB problems: Each ciphertext block adds randomness to encryption of following block
- ❌ Propagates errors and no parallel encryption

Counter mode (**CTR)**

- ✅ Uses a nonce and an increasing counter to introduce randomness across ciphertext blocks
- ✅ Parallel encryption and decryption

61

Summary: Block ciphers

STRENGTHS

High diffusi …ed into several ciphertext sy…

Immunity to …ection

WEAKNESSES

Slow: an ent… decryption can begin

Error propag…

Don't design your own

***AES* – The Advanced Encryption Standard**
128/256 bit key, NIST Standard, HW support

More: https://en.wikipedia.org/wiki/Block_cipher#Notable_block_ciphers

Computer Security and Privacy (COM-301)

Applied cryptography

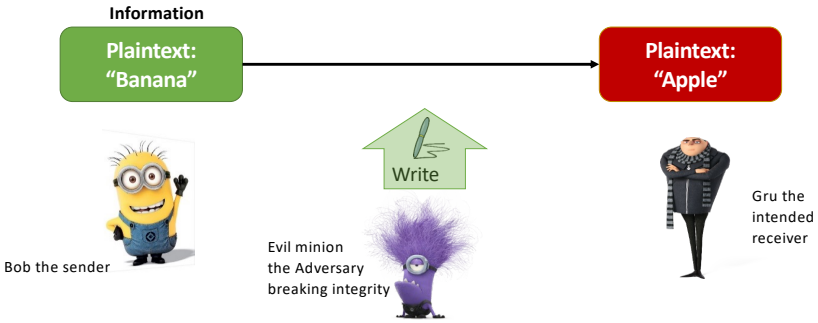
Symmetric encryption - Integrity

Carmela Troncoso
SPRING Lab
carmela.troncoso@epfl.ch

Some slides/ideas adapted from: George Danezis, Yoshi Kohno

Cryptography for integrity

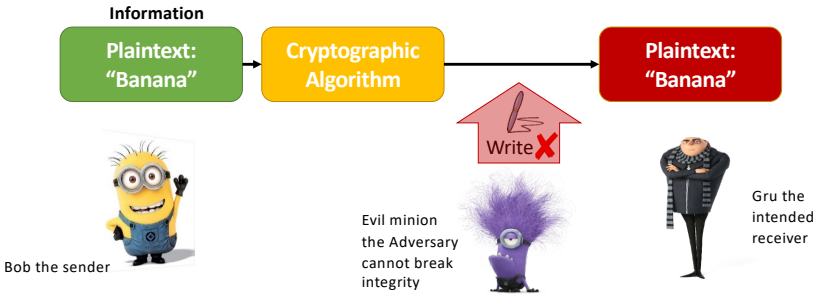
Integrity: information cannot be modified by unauthorized parties



Recall that integrity means that non-authorized parties cannot modify data. That is if Bob sends "Banana" he can be sure that Gru will receive "Banana".
Without cryptography, it is very easy to violate integrity.

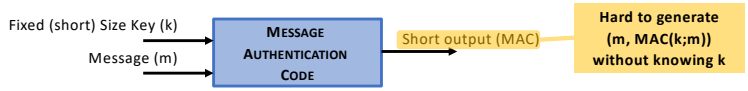
Cryptography for integrity

Integrity: information cannot be modified by unauthorized parties



In the next part, explain how we can use cryptographic algorithm to ensure integrity.

Integrity from symmetric encryption: Message Authentication Codes (MAC)



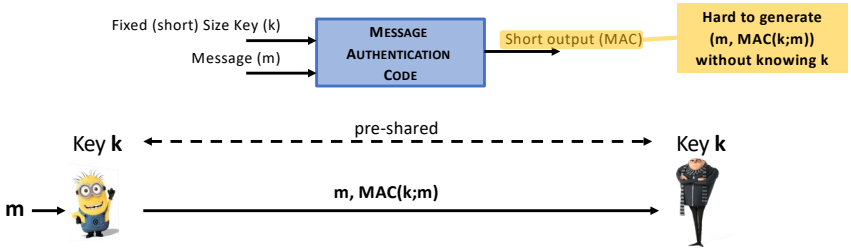
A **Message Authentication Code** receives two inputs:

- A small key that must be kept secret
- The message whose integrity needs to be secured

The MAC algorithm outputs a short string that helps the intended receiver to verify the integrity of the message sent by the sender.

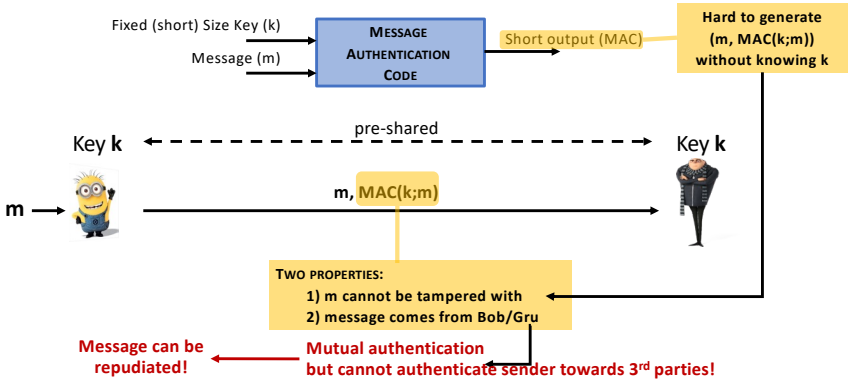
The key property of the MAC is that *it is very difficult to produce a pair [message; MAC(k,message)] without knowing the key*

Integrity from symmetric encryption: Message Authentication Codes (MAC)



To verify the integrity of the message, Gru inputs the message and the key to the MAC algorithm, and compares the output to the MAC he received. If they are the same, the message has not been tampered with.

Integrity from symmetric encryption: Message Authentication Codes (MAC)



Because only Bob and Gru know the key, when receiving a message with a valid MAC they know that only the other could have written it. However, they cannot prove to a third party that they are not the author of the message (as they know the key, they could have produced the correct MAC). Thus, the message can be *repudiated*. Bob can say that the message was written by Gru; and Gru can say that the message was written by Bob.

Example MAC: CBC-MAC

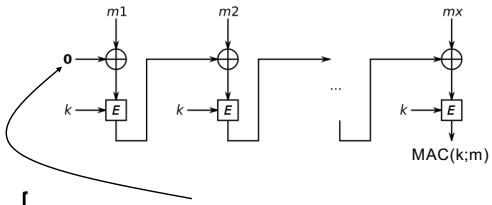
Turning a block cipher into a MAC

Limitation:
Only secure if the length of m is known!

$C_0 = 0$ **[any fixed IV]**

$C_i = Enc(k; m_i \oplus C_{i-1})$

$MAC(k; m_1 ... m_x) = C_n$



Differences with respect to CBC

- **CBC-MAC** is deterministic
- Only output is the final value!

A block cipher in CBC mode can be turned into a MAC by:

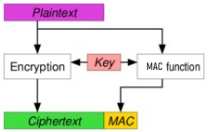
Fix the IV (for example 0) and then do CBC. The MAC is the output of the last encryption block: one could have only gotten there for one message and one key. Note that the last block cannot be used to recover anything about the message: it looks random (as it is the output of a block cipher)

As the IV is fixed, the result for one message is always the same (no value changes!)

It is only secure (ensures integrity) if the length of m is known. This is because the length of the message determines the output of which block is the MAC. But, if Gru does not know the length of the message, it is easy to get a MAC for an extension of the message:
If you have T= CBC-MAC(k,M), and T'=CBC-MAC(k,M'); then T' is a correct CBC MAC for M || T XOR M' where || is concatenation.

How to obtain confidentiality and integrity?

ENCRYPT-AND-MAC



✗ No integrity check on the ciphertext →
Cipher can be attacked
Need to decrypt to know if message is valid

✓ Integrity of the plaintext can be verified

✗ May reveal information about the plaintext →
Repeated msg, recall the IV of the MAC is fixed
(can be solved with a counter)

Bellare, M., & Namprempre, C. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. *International Conference on the Theory and Application of Cryptology and Information Security*, 2000.
Bellare, M., Kohno, T., & Namprempre, C. Breaking and provably repairing the SSH authenticated encryption scheme: A case study of the Encode-then-Encrypt-and-MAC paradigm. *ACM Transactions on Information and System Security*, 2004.
https://en.wikipedia.org/wiki/Authenticated_encryption

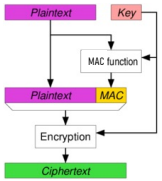
How to obtain confidentiality and integrity? Encrypt-and-MAC?

- The MAC is computed on the plaintext, so it cannot protect the integrity of the ciphertext. To check whether the message is correct, one needs to first decrypt (expensive operation).

- Because the MAC is deterministic, it may reveal information about the message (e.g., if the message is sent twice the MAC is the same for both messages)

How to obtain confidentiality and integrity?

MAC-THEN-ENCRYPT



✗ No integrity check on the ciphertext →
(In theory) possible to change ciphertext and have a valid MAC
Need to decrypt to know if message is valid

✓ Integrity of the plaintext can be verified

✓ No information on the plaintext since it is encrypted

Bellare, M., & Namprempre, C. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. *International Conference on the Theory and Application of Cryptology and Information Security*, 2000.
Bellare, M., Kohno, T., & Namprempre, C. Breaking and provably repairing the SSH authenticated encryption scheme: A case study of the Encode-then-Encrypt-and-MAC paradigm. *ACM Transactions on Information and System Security*, 2004.
https://en.wikipedia.org/wiki/Authenticated_encryption

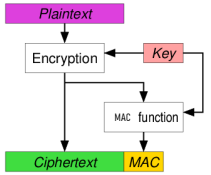
How to obtain confidentiality and integrity? MAC-then-Encrypt?

- Still no integrity check on the ciphertext

- But no info on the MAC because the adversary does not see it (only sees the encrypted blob)

How to obtain confidentiality and integrity?

ENCRYPT-THEN-MAC



- ✓ Integrity of ciphertext →
  - Ensures you only read valid messages
  - Cipher cannot be attacked
- ✓ Integrity of the plaintext can be verified
- ✓ No information on the plaintext since it is encrypted

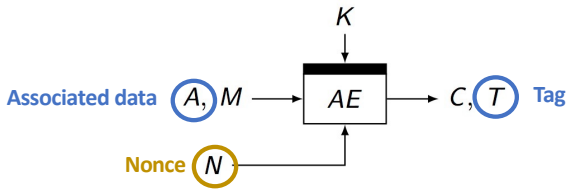
Bellare, M., & Namprempre, C. Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. *International Conference on the Theory and Application of Cryptology and Information Security*, 2000.
Bellare, M., Kohno, T., & Namprempre, C. Breaking and provably repairing the SSH authenticated encryption scheme: A case study of the Encode-then-Encrypt-and-MAC paradigm. *ACM Transactions on Information and System Security*, 2004.
https://en.wikipedia.org/wiki/Authenticated_encryption

How to obtain confidentiality and integrity? Encrypt-then-MAC?

Ensures integrity of cipher- and plaintext. No information about plaintext.

In practice… (out of the course scope)
Authenticated Encryption with Associated Data (AEAD)

New constructions to avoid home-made combinations



Galois counter mode - **GCM** (one pass)

Encrypt-then-authenticate-then-translate - **EAX** (Two passes)

https://www.fi.muni.cz/~xsvenda/docs/AE_comparison_ipics04.pdf

The combinations are easy to confuse and result in problems. Newer schemes provide *authentication and confidentiality in one go*. For this they require, besides the key and the Nonce/IV more information called associated data.

The output of **Authenticated Encryption** is the ciphertext and a **Tag** that can be used to check integrity.