TOPIC 2
CRYPTOGRAPHY APPLICATIONS

Protecting your data from leaks & unauthorized access

JOSÉ MANUEL REDONDO LÓPEZ *"S-81 ISAAC PERAL"* PROJECT V4.0




Source: Bing Chat AI

Departamento de Informática
Universidad de Oviedo

Universidad de Oviedo

Logo: @creative_vanesa (*Instagram*)

INDEX

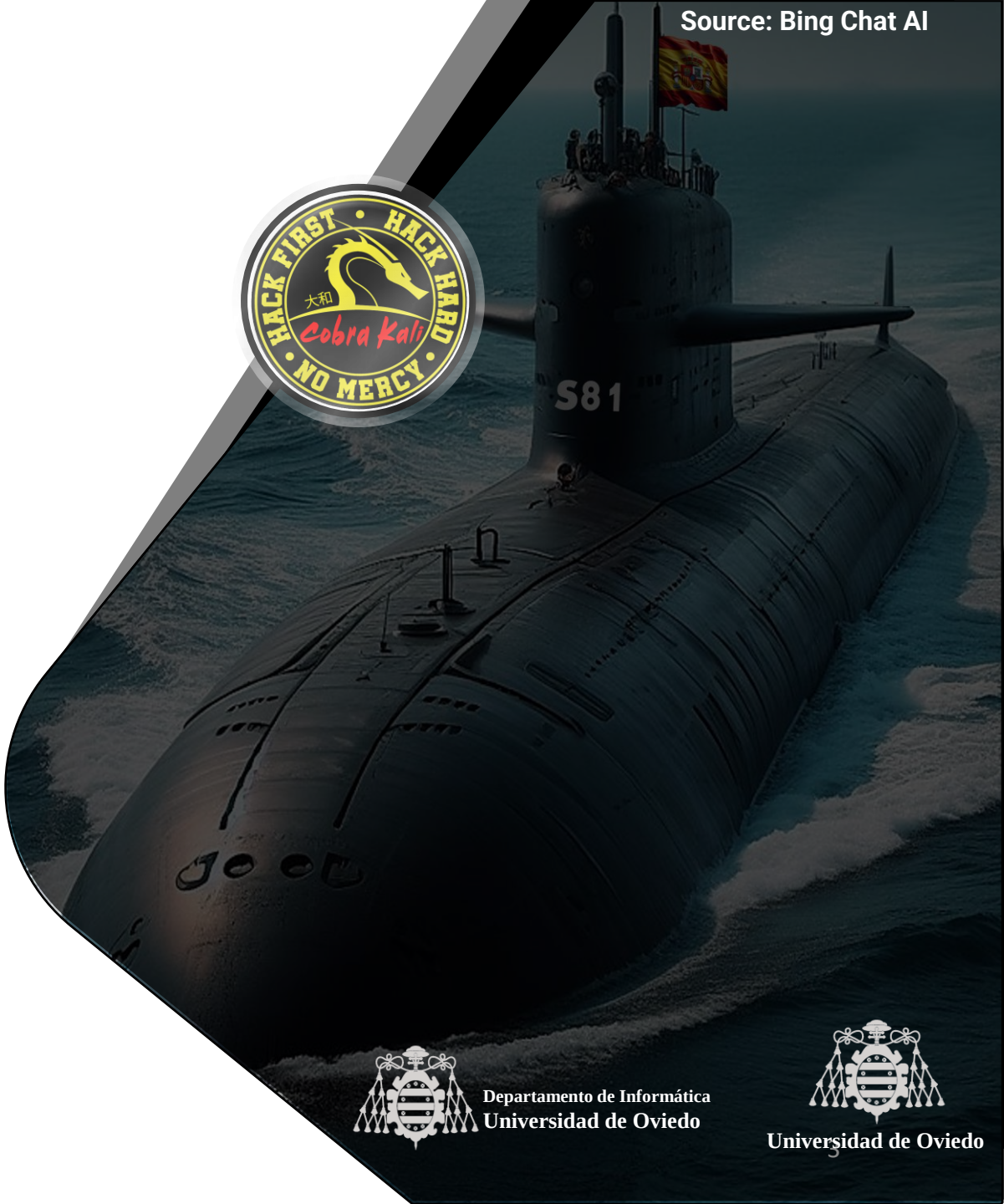
< Go to Index

INTRODUCTION

What are we talking about?

*Icons by Bing Chat AI

Source: Bing Chat AI



What are you going to find in this block?

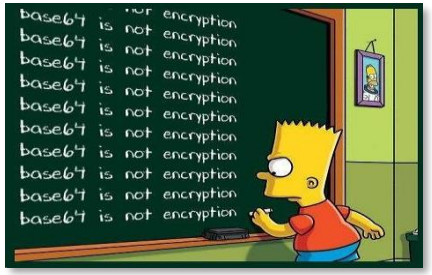
- **In this block you will learn**
  - What are a series of operations that manipulate (but do not encrypt data): **minification**, **encoding**, and **obfuscation**
    - And the differences between them
  - What **cryptography** and **cryptanalysis** are
  - The difference between cryptography and **steganography**
  - Why steganography **does not guarantee security** alone
  - What is the "**CIA**" when related to cryptography

General topic goals

- **Know the foundations and applications of the modern different cryptography techniques**
  - How and why they are applied in different scenarios
  - Why some techniques are more adequate than others to provide certain security characteristics
- **Eliminate false beliefs**
  - **Code minification** and **file obfuscation** are **NOT** cryptography techniques
    - Programs are still executable, only obscures the meaning of the code
  - **File/data encoding** is also NOT a cryptography technique
    - E. g.: MIME (https://en.wikipedia.org/wiki/MIME), Base64 (https://es.wikipedia.org/wiki/Base64)…
    - Only allows the contents to be transferred with a character set compatible with the transmission protocol
    - It is easily reversable

Minified JavaScript. Source:
https://dev.to/aryaziai/minifying-code-shortcut-4d6c

```
{font-family:sans-serif;-ms-text-size-adjust:100%;-webkit-text-size-
}body{margin:0}article,aside,details,figcaption,figure,footer,header,hgroup,mai
parent}a:active,a:hover{outline:0}b,strong{font-weight:bold}h1{font-size:2em;ma
,input,optgroup,select,textarea{color:inherit;font:inherit;margin:0}button{over
"button"],input[type="reset"],input[type="submit"]{-webkit-appearance:button;cu
r:0;padding:0}input{line-height:normal}input[type="checkbox"],input[type="radio
```




```
This is an MIME test email
--VGhpc0lzQW5FbWFpbEJvdW5kYXJ5Cg==
Content-Type: application/vnd.openxmlformats-officedocument.spreadsheetml.sheet
Content-Transfer-Encoding: base64

Q29udGVudC1UeXBlOiBtdWx0aXBhcnQvbWl4ZWQ7CiAgICBib3VuZGFyeT0iMTI3LjAuMC4xLjUw
MS4xNjkyMy4xNTIyNDA4ODQ3Ljc1My4xIgpNaW1lLVZlcnNpb246IDEuMAoKaWYgeW91IGNhbiBy
ZWFkIHRoaXMsIHlvdXIgbWFpbGVyIGIzIG5vdCBNSU1FLWF3YXJlCgotLTEyNy4wLjAuMS41MDEu
MTY5MjMuMTUyMjQwODg0Ny43NTMuMQpDb250ZW50LVR5cGU6IHRleHQvcGxhaW4KQ29udGVudC1U
```

An Excel file sent as an email attachment, using Base64 encoding. Source:
https://stackoverflow.com/questions/49605212/failed-to-send-an-mime-email-body-with-mail-command

➢ Exercise **L3B1_ENCODE, L3B1_OBFUSCATE**

CRYPTOLOGY

- **Science that deals with secure information exchange problems**
  - It always uses an information **source** and **destination**, along with a **communication channel**
  - Aims to generate highly-secure, “unbreakable” communications
    - Breaking them means reading their contents without proper authorization
  - It is very old! (2000+ years)
- **Two main research lines**
  - **Cryptography**: systems able to **hide information meaning**
    - Generating ciphered information inaccessible to those not meant to know it
  - **Cryptoanalysis**: systems able to **decipher** ciphered information
    - Break a cryptography system

Source:
https://en.wikipedia.org/wiki/Enigma_machine

Source:
https://danbrown.fandom.com/wiki/Cryptex

WHY CRYPTOGRAPHY?

- The main goal of a secure system is to **achieve proper protection of the data / information it manages**
- To do that we can use two techniques
  - **Cryptography**: hide the **meaning** of the information
    - Information is processed and transformed into a **non-readable version** (cyphered / encrypted)
    - This is achieved by processing the information through different **encryption algorithms**
    - The chosen algorithm depends **on the purpose of the information** (private data, passwords…)
    - **Modern cybersecurity greatly depends on using appropriate cryptography methods**
  - **Steganography**: hide the **existence** of the information
    - Information is **embedded** into another different piece of information (e. g. image)
      - Is not accessible to those that do not know that it exists (obscure its presence)
    - However, using **only security by obscurity is not an advisable technique!**
      - Normally, you combine it with cryptography

Source: *PowerPoint* icons

STEGANOGRAPHY

Message to hide

<secret>
This image is less innocent than it looks. In fact, it's hiding this secret message (you know it is secret because it is enclosed between <secret> </secret> tags ☺)
</secret>

"Container" file

Password

Steganographic process

Steganogram: The image
Now contains more information (the message), but you cannot see any difference at plain sight!

Source: https://www.incibe-cert.es/blog/esteganografia-y-estegoanalisis-basicos

STEGANOGRAM GENERATION

Seguridad de Sistemas Informáticos

- Steganograms are generated (and extracted) with appropriate tools
- Some of the most popular are
  - Steghide
    - steghide.sourceforge.net/documentation/manpage_es.php
  - Steganos Security Suite
  - Outguess
  - Stegtools
  - SilentEye
  - Zsteg
    - https://github.com/zed-0xff/zsteg
  - **For images**
    - Exiftool: https://en.wikipedia.org/wiki/ExifTool
    - Stegsolve: http://www.caesum.com/handbook/Stegsolve.jar

Source: https://www.publicdomainpictures.net/es/view-image.php?image=215827&picture=mujer-de-yoga

Extract a hidden message with `steghide`

```
root@kali:~/Descargas# steghide extract -sf lady-in-yoga.jpg
Anotar salvoconducto:
anot? los datos extra?dos e/"secret.txt".
root@kali:~/Descargas#
```

```
root@kali:~/Descargas# cat secret.txt
Don't forget to knock: 3000,3001,3002
root@kali:~/Descargas#
```

The image appears to be normal and visually there is no difference, but it contains a message inside

```
root@kali:~/Descargas# exiftool lady-in-yoga.jpg
ExifTool Version Number         : 11.65
File Name                       : lady-in-yoga.jpg
Directory                       :
File Size                       : 21 kB
File Modification Date/Time     : 2019:10:07 13:02:12+02:00
File Access Date/Time           : 2019:10:07 13:02:20+02:00
File Inode Change Date/Time     : 2019:10:07 13:02:12+02:00
File Permissions                : rw-r--r--
File Type                       : JPEG
File Type Extension             : jpg
MIME Type                       : image/jpeg
JFIF Version                    : 1.01
Resolution Unit                 : None
X Resolution                    : 1
Y Resolution                    : 1
Image Width                     : 283
Image Height                    : 676
Encoding Process                : Baseline DCT, Huffman coding
Bits Per Sample                 : 8
Color Components                : 3
Y Cb Cr Sub Sampling            : YCbCr4:4:4 (1 1)
Image Size                      : 283x676
Megapixels                      : 0.191
root@kali:~/Descargas#
```

STEGANOGRAPHY + CRYPTOGRAPHY

- **Just hiding a message is not a trustable security method**
  - **"Security by obscurity" is NEVER a good idea!**
- **For that reason, steganography is combined with cryptography**
  - The hidden message to be exchanged must be encrypted before placed in the container object
  - Even the steganographic pattern is discovered, the message remains unknown
- **We obtain a significant advantage**
  - If only encryption is used, we cannot see the meaning, but we know that something is transmitted
    - Therefore brute-force attacks (or other variants) can be used against the encryption algorithm (against that "something" being transmitted)
  - If both are used, the probability of transmitting information undetected increases
    - You cannot attack something that you are not aware of! ☺
- **Let's see the ciphering forms we can use for that!**

CRYPTOGRAPHY OBJECTIVES

- **Encrypted communications are basic to achieve real security**
  - With or without hiding the message transfer!
- **Cryptography aims to attain this (colloquially named the "CIA")**
  - **Confidentiality**: Only those knowing the deciphering method may know the information meaning
    - This applies to stored information as well as network traffic
    - **Ciphering algorithms** are used for this
  - **Integrity**: Assurance that the transmitted data were received exactly as the sender sent them
    - **Without any alteration / tampering**: information remains unaltered from source to destination
    - **Hashing functions** are used for this
  - **Authentication**: Verification that the identity of senders and receivers is the declared one
    - You don't want to sent private data to / accept it from untrusted / fake entities!
    - This applies to users and machines
    - **Digital signatures and certificates** are used for this (**metadata** and **watermarks** are not trustable)
- **Authenticated encryption algorithms (seen later) provide all three objectives**
  - https://en.wikipedia.org/wiki/Authenticated_encryption

➢ Exercises **L4B4_WATERMARK, L4B4_META**

Think you've understood it all? Evaluate yourself!

?

- **You can do the following**
  - *Understand that minifying, encoding, and obfuscating have their uses, but they ARE NOT encryption*
  - *Understand that, if minified code takes up less space, apart from being less readable, it improves the loading performance of websites*
  - *Be aware that cryptography (and its "opposite", cryptanalysis) are very old mathematical sciences*
  - *Be aware that the essence of steganography, regardless of the software or method used to implement it, is to send information without anyone knowing it, and therefore without it being obvious to the naked eye*
  - *Understand that steganography without cryptography does not guarantee security, because hiding information alone is never safe*

< Go to Index

Source: Bing Chat AI

Symmetric >

Asymmetric >

CONFIDENTIALITY

Ciphering algorithms

*Icons by Bing Chat AI

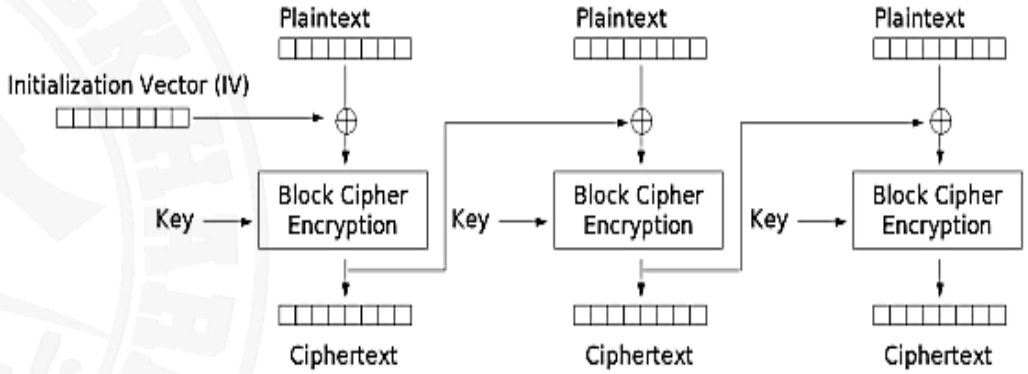
WHAT ARE YOU GOING TO FIND IN THIS BLOCK?

In this block you will learn

- The two types of algorithms that **use cryptographic keys** and other types of encryption
- The basic principles of **cryptographic keys**
- The theoretical basis **of symmetric-key algorithms**, their drawbacks, and some of their most popular applications
- the theoretical basis of **asymmetric key algorithms**, their drawbacks, and some of their most popular applications

Classification of ciphering algorithms: how they process the information

- (De)ciphering algorithms are **functions** that **transform** information
  - **Ciphering** hides the meaning of the information
  - **Deciphering** recovers it back
    - If applied over **a compatible ciphering algorithm**
- Ciphering algorithms may be classified in
  - **Block ciphers**: IDEA, AES, RSA …
    - They cipher information in chunks (blocks)
    - Block size (in bits) varies with the algorithm: 64, 128…
    - Use a random **Initialization Vector** (IV)
      - Data to initialize ciphering
      - How they use this IV determines its **operation mode**
        - ECB, CBC, PCBC, OFB…
      - The mode could be part of the algorithm name. E. g.: AES-CBC
      - Algorithm modes will not be reviewed
  - **Stream ciphers**: A5, RC4, SEAL …
    - Ciphers information at the single bit level



Cipher Block Chaining (CBC) mode encryption

Source: https://es.wikipedia.org/wiki/Modos_de_operaci%C3%B3n_de_una_unidad_de_cifrado_por_bloques

Classification of ciphering algorithms: How they use keys

- **Keyless ciphers** (aka **transposition ciphers**)
  - Positions of units of plaintext are **shifted** using a regular system
    - Units of plaintext are commonly characters or groups of characters
    - Ciphered text is a permutation of the plaintext (**plaintext is reordered**)
  - A mathematically bijective function is used on the characters' positions to encrypt (and an inverse function to decrypt)
  - **Examples**: ROT13, ROT47, … (test them in https://cryptii.com/)
- **Cryptographic key ciphers** (**the ones studied in the course**)
  - Use **keys**: key owners can manipulate the information
  - **Symmetric-key algorithms**
    - Sender and receiver **share** the same private key
  - **Asymmetric algorithms**
    - **Sender and receiver have two keys each**
    - **One is Public** (to cipher)
    - **The other Private** (to decipher / authenticate)
    - **The base of modern e-commerce!**

	SYMMETRIC	ASYMMETRIC
Key type	Shared secret key	Unique private & public key pair
Strengths	• **Fast performance** • Easy to understand (having the key means you can access the information)	• **Private keys are never exposed** • Also provides authentication of the source
Weaknesses	• Shared secret (exposed key) • Does not authenticate the source (if the key is exposed, information can be forged)	• Public-key management requires additional infrastructure (CAs, Trusted Rings) • **Computationally intensive** (compared with symmetric)
Key differences	Applications must store keys that, if found, **may enable forgery**	**Resistance to forgery** if private keys are kept secret

CRYPTOGRAPHIC KEY CIPHERS

- The **Kerckhoffs principles** are the main guidelines to create an unbreakable key-based ciphering algorithm
  - Enunciated in 1883!
  - **Effectiveness must not depend on the secrecy of the algorithm design**
    - The **key must be the most important element** to successfully decipher a ciphered message
    - Even if the algorithm is public, the information cannot be deciphered without an adequate key
  - **Keys should be easy to remind**
    - To avoid writing them in non-secure storage media
  - Resulting ciphered information must be **alphanumeric**
  - Ciphering systems must be **operable by a single person**
  - Ciphering systems should be **easy to use**
- Modern cryptography is influenced by Claude Shannon's information theory
  - https://en.wikipedia.org/wiki/Information_theory

CRYPTOGRAPHIC KEY

- Information **pattern that cryptographic algorithms use to cipher and decipher the information**
- **We will obtain a totally different output if we change the key**
  - Even if using the same algorithm and starting data!
- **The key strength is measured in bits or digit length**
- **For a given algorithm, longer keys means stronger keys**
  - However, longer keys require more time to process the information
  - Strong means **difficult to break**
    - I. e. decipher the information without having the key

```
Original_information = Decipher_Algorithm (Cipher_key,
        Cipher_Algorithm (Original_information, Cipher_Key) )
```

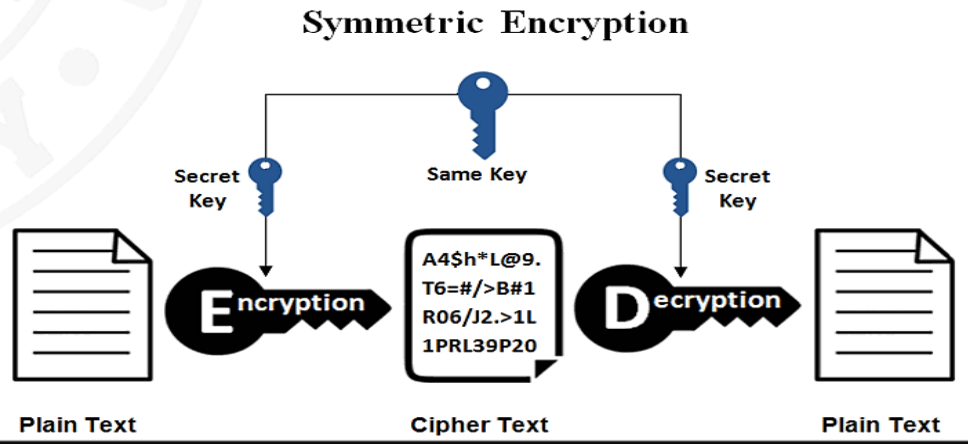
Symmetric-key algorithms

Ciphering with only one shared key

*Icons by Bing Chat AI

SYMMETRIC-KEY ALGORITHMS (AKA PRIVATE KEY ALGORITHMS)

- **The deciphering function is the inverse of the ciphering one**
  - These algorithms are designed to maximize the "**avalanche effect**"
    - Just changing a single bit in the input produces a great change in the output (the more, the better)
  - Mathematical functions are used to make deciphering more difficult
    - Non-linear functions, XOR functions…remember the CN and Algebra courses
    - They are designed using a minimal "core" of functions, that are repeated several **rounds**
    - Rounds are parametrizable, increasing security but also computational cost (more rounds -> more CPU)
- **Sender and receiver share the same key K**
  - This key is not internally used "as is", but "subkeys" are usually generated by the algorithm
    - This favors the "avalanche effect"
  - This has the **key distribution problem**
    - Senders and receivers must share K
    - So, it **must be transmitted, protected, and stored securely**
    - Anyone with a copy of K can decipher all information!



Source: https://cryptobook.nakov.com

SECURE SYMMETRIC-KEY ALGORITHMS

- An algorithm is considered **computationally secure** if an attacker cannot break its encryption faster than a brute-force attack
  - It would use more time that testing all possible keys!
  - Which can take millions of years, depending on key size…
- Some examples of current **block-cipher** algorithms considered secure are
  - **AES (Advanced Encryption Standard)** (2001)
    - Adopted as an **effective standard of encryption** by the U.S. government in 2006, replacing DES and 3DES
    - Divides the data into 128-bit blocks, and uses 128-bit, 192-bit, or 256-bit keys
    - Has multiple **operation modes**: AES-GCM-SIV, AES-GCM, AES-CTR, and AES-CBC
    - Currently there is no known practical attack allowing someone without the key to read encrypted data, when AES is correctly implemented (**strong cipher**)
  - **IDEA (International Data Encryption Algorithm) NXT**
  - **Twofish**
  - **Serpent**

AES is used everywhere for symmetric ciphering

PayPal

Technical Details
Connection Encrypted (TLS_AES_128_GCM_SHA256, 128 bit keys, TLS 1.3)
The page you are viewing was encrypted before being transmitted over the Internet.

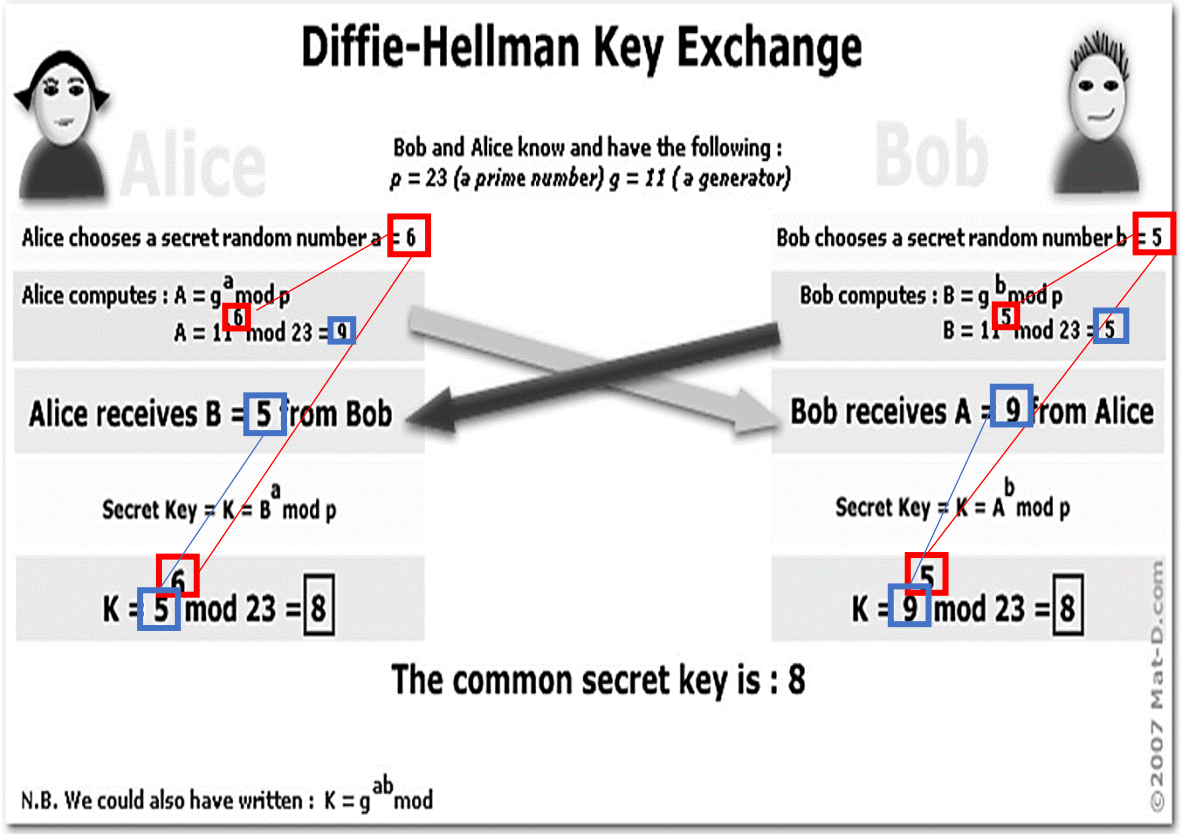
Google

Technical Details
Connection Encrypted (TLS_AES_128_GCM_SHA256, 128 bit keys, TLS 1.3)
The page you are viewing was encrypted before being transmitted over the Internet.

➢ Exercise **L3B2_SYMCRYPT**, **L3B2_SYMDECRYPT**, **L3B2_CIPHMODES**

Solving the key distribution problem: Diffie-Hellman key exchange algorithm

- **Establishes a symmetric key between 2 machines to cipher communications**
  - Can be used in **insecure communication channels**
  - The resulting key cannot be obtained by an attacker, even if the messages are captured!
- **Uses the following steps**
  - One machine chooses two numbers (p, g) and send them to the other machine in a **public** way
  - Each one chooses another **secret** number <p
  - Using a mathematical formula, each machine perform a series of operations
    - **With the two public numbers and the secret one**
  - This is the secret key used to cipher the message
- **Reverting the calculation to obtain the common secret key is very expensive**
  - **If the key is big enough**: **secure algorithm**



Source: https://stackoverflow.com/questions/28015433/is-it-possible-to-hack-diffie-hellman-by-knowing-the-prime-number-and-the-gene

How big is "big enough"?: Real implementations uses 2048-bit numbers (in 2016, the NSA recommended 3072+ bits)

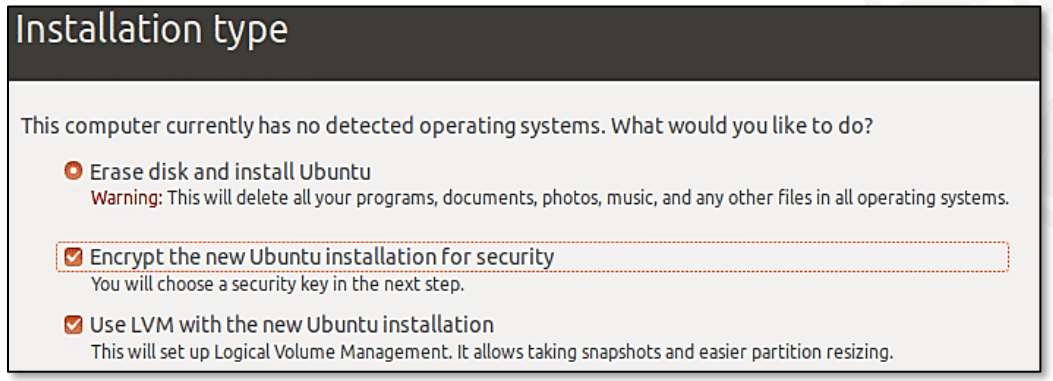
http://www.slideshare.net/samehelhakim/introduction-to-vpn-47256821

Solving the key distribution problem: Forward Secrecy

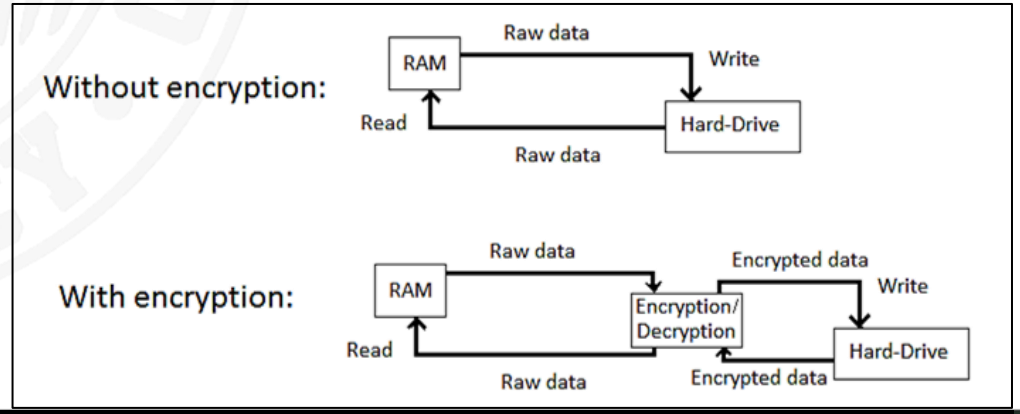
- **It is a property of the algorithms that must exchange keys that provides more security to communications in the following way**
  - Imagine that you save a large amount of encrypted traffic from a communication between two entities
  - Now imagine that you somehow **get the symmetric encryption key used to make it**
  - You'd have access to **all future communications**... and to the past ones too!
- **If the encryption algorithm has the property of forward secrecy, a compromise of an encryption key only affects that compromised communication session**
  - In other words, this key **cannot be used to decipher past** (or future) traffic between the two entities
  - Because **a unique symmetric key** is generated to be exchanged for each communication session
  - That implies that you must be careful with the symmetric algorithm you choose
    - Not because it's weak, but because of the impact of an encryption key compromise
  - For example, **GPG** (used in many encrypted email systems) **does not have it**
    - If your encryption key is compromised, the attacker could read all your email ☹
    - https://alexgaynor.net/2017/apr/26/forward-secrecy-is-the-most-important-thing/
  - **OpenSSL** (the library that implements the encryption used on the Internet) **does**

Symmetric-key algorithms practical applications: Disk encryption

- **Technology to cipher stored information, preventing unauthorized access**
  - Preventing data leaks due to physical hard-drive theft is the most common usage example
  - Data of an encrypted volume cannot be read (decrypted) without the correct password
- **Uses symmetric-key encryption software / hardware to encrypt every bit**
  - The entire file system in a volume or disk is encrypted
  - File / folder names, contents, and metadata
- **It is commonly transparent**
  - Also called real-time encryption or on-the-fly encryption (OTFE)
  - Data is automatically encrypted or decrypted as it is loaded or saved
  - Users notice no difference with handling unencrypted data

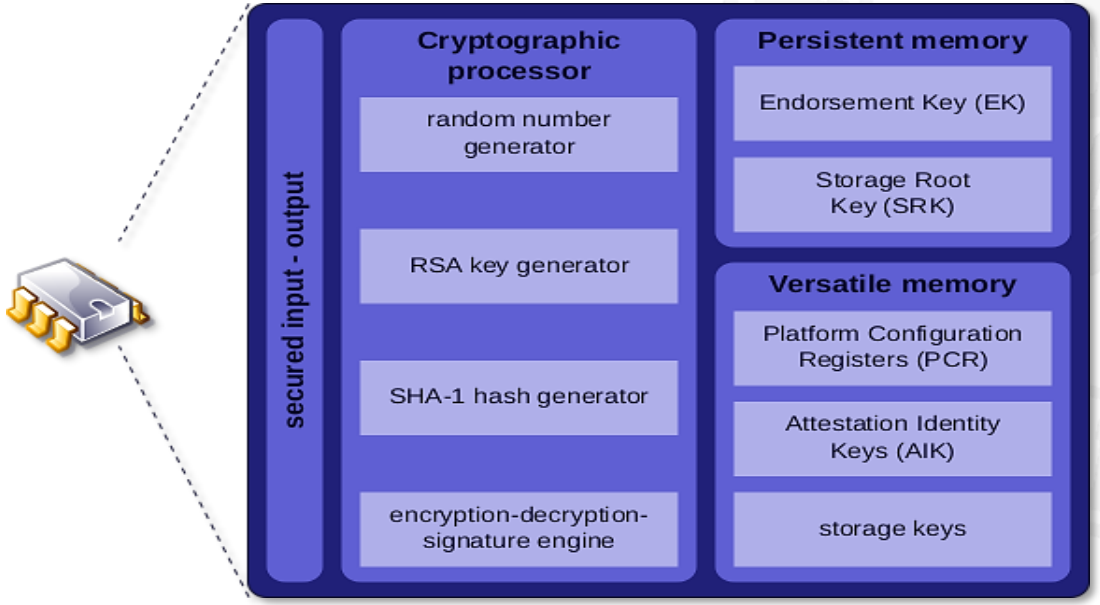


Source:
https://superuser.com/questions/448965/does-full-disk-encryption-on-ssd-drive-reduce-its-lifetime

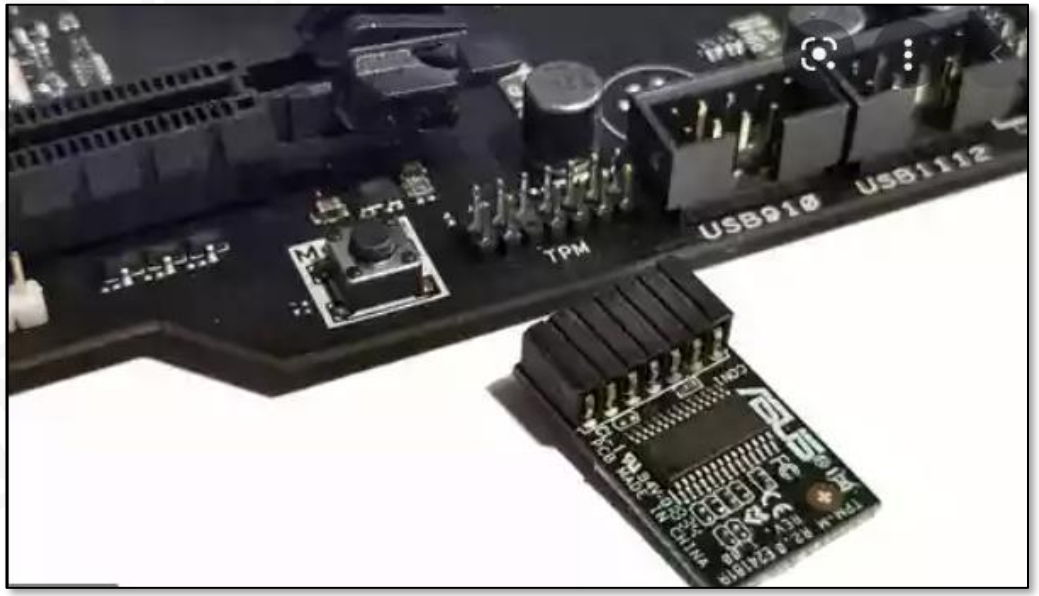


Symmetric-key algorithms practical applications: Platform authentication

- **Trusted Platform Module (TPM)** is a secure crypto-processor embedded in some motherboards (or CPUs) that can be used to authenticate a hardware device
  - Each TPM chip is unique to a device, so it can perform platform authentication
  - It can verify if a system accessing a device is authorized
    - "Transplanting" an encrypted drive to read it does not work!
  - It is also a requisite to install Windows 11 nowadays



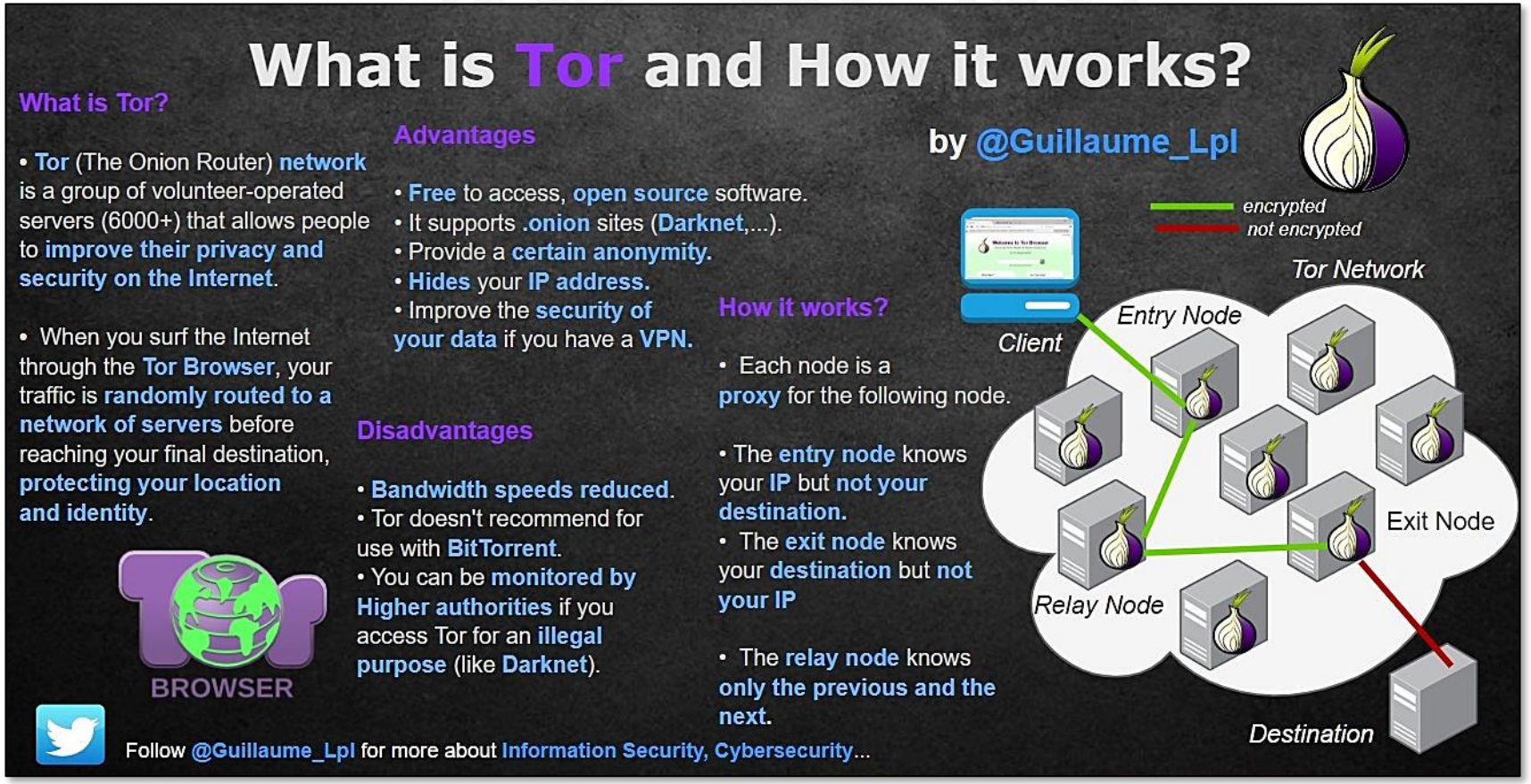
Source: https://sergioprado.blog/introduction-to-tpm-trusted-platform-module/



Source: https://hardzone.es/reportajes/que-es/trusted-platform-module-tpm/

Symmetric-key algorithms practical applications: Onion routing (encrypted anonymous transmissions)

- The **ToR** anonymization network nodes exchange symmetric keys (AES-128 in 2021) between them using **Diffie-Hellman**
  - https://blog.insiderattack.net/deep-dive-into-tor-the-onion-router-6de4c25beba7?gi=2acdf11e842c
  - https://linuxconfig.org/install-tor-proxy-on-ubuntu-20-04-linux



Asymmetric (public key) algorithms

Ciphering with a pair of keys per user

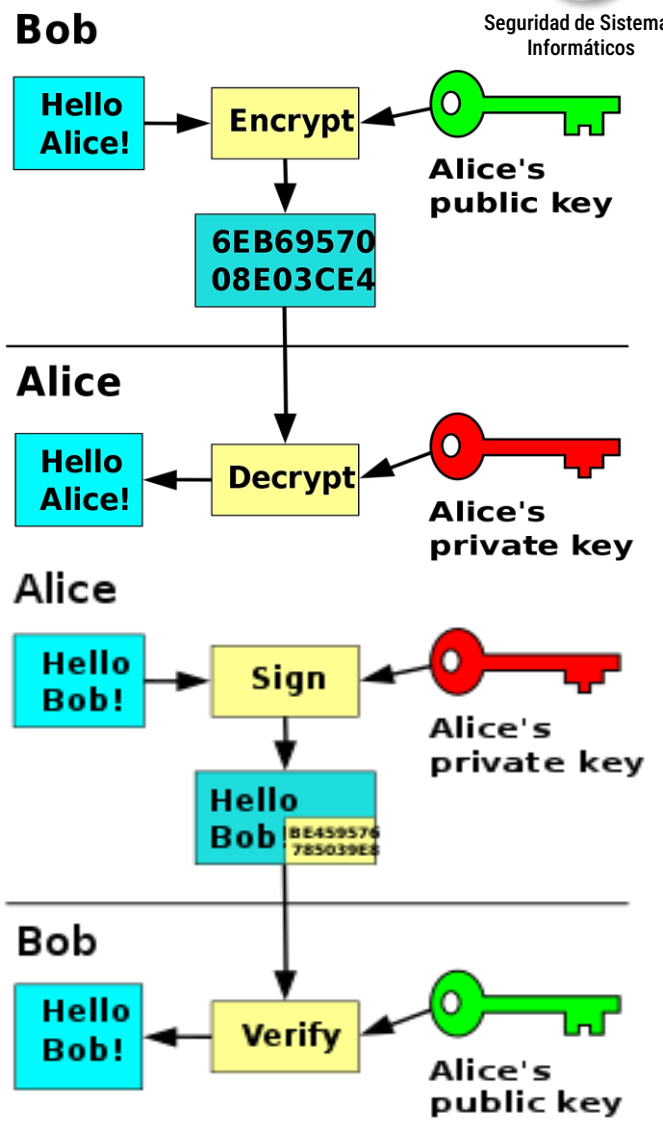
*Icons by Bing Chat AI

Asymmetric (public key) ciphers

- **Avoid the key exchange problem of symmetric key ciphers**
  - Senders and receivers do not need to agree on a key, each have its own **key pair**
  - **The keys of each pair (DK, EK) are generated together, not individually**
  - **Each pair is unique for each communication party**
  - One key of the pair is **private** (DK, decryption key) and **must be kept secret**
  - The other key of the pair is **public** (EK, encryption key) and **is distributed to all potential recipients**
- **This way**
  - If a user wants to **receive encrypted messages**, it sends the **public** key EK to the sender
    - Or the sender gets it from the receiver
    - And **privately stores its DK decryption key**
  - Information **encrypted with the private key can only be decrypted with the corresponding public key** of its pair, **and vice versa**
  - **Remember**: a particular key pair **can only be generated once**
  - **The private key cannot be inferred from its corresponding public key in the pair**
    - There is no danger in transmitting public keys over the network

Asymmetric (public key) algorithms

- **Bob uses Alice's public key to send her encrypted information**
  - He then will be sure that the recipient is indeed Alice
  - The information can only be deciphered with Alice's private key
    - If Alice's private key decrypts the received information, it must have been encrypted by her public key
    - Encrypted messages for user U1 cannot be read (decrypted) by user U2
      - Unless U2 steals U1's private key!
  - Opposite, data encrypted by Alice's private key can only be decrypted by Alice's public key
    - Hence authenticating its sender
    - If Alice's public key can be obtained in a trustable way
- **RSA and ElGamal are popular algorithms of this type**



Source: https://en.wikipedia.org/wiki/Public-key_cryptography

CRYPTOGRAPHIC KEYS AND USER PASSWORDS ARE NOT THE SAME!

- A cryptographic key **IS NOT a password**
- Ex.: An **AES symmetric key** is an hexadecimal string of variable length
  - Depending on the number of bits of the key (strength)
    - https://www.ibm.com/docs/en/imdm/12.0?topic=encryption-generating-aes-keys-password
  - Ex.: AES-256
    - `salt=D495560961CCCFE0` (**text that is added** to the ciphertext to ensure a random output even if the input is the same)
    - `key=4D92199549E0F2EF009B4160F3582E5528A11A45017F3EF8` (the key itself)
    - `iv=35B2FF0795FB84BBD666DB8430CA214E` (A random value used to generate the key)
- **Asymmetric key pairs** can be written in plaintext (armored format) (see images)
  - As you can see, they are **BIG** ☺

```
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: BCPG v1.38

mQGiBEk5VoARBAC69sz9r6gm6fG7Y1rC4xxLOBVeQfTzp/3vXQGfnVOE13+1MNye
aQ4CITZ1/955HdHZvoDcF/IXEr+vhT7gMEMyHfGOdOaSQVJ2ObzeyTOYjrkOvzP6
5zJIOceJICzTsB1FsnWj5DZpTU4ycidc9cV+/lx1aiJLjXXShCqbMNq4RQCggx9m
6GxGERCzxP1rT2AyQpGoUdcykwMDAgo+Q1c6XYz7jVUQJuNZCYQoDovO7SbDW8Tu
O9/ZvEiU36YxqZw953K4zyVvlY8PB4f4AqlbO/MRQv14LG93OrreXmOa8c3iEBLg
Y1u8JYnEL3Tmkrb6xWfseDxaWwYsbvlD5QQAjfcazqjGpTxWYkaxb5WGeDu4Abzs
sDmYq1qGAAvMtAJFu5sW8FO+SahSdfAOpIyOaX6unlUXXIw6NoDI8vQLYGKjM7Om
sXwqlucY5SO/H+41PrV71LrHBWw+Xdutb+pPnFm61Q5unUxItkfYOOtgadDQvNuo
mco93MZv5AOmvzMEALl87JHmhmIOOfq8jZGfl1eWyeCz4SwNBgHBvC/tjWcKjpZE
fak1upvELMbjGvneQaXJtAXGnCUfDOuhZ8rBOkC9G/7EQJCmFy5iAIiSE2+NsJTZ
gn8K8swVEQxTcOk4y3ARP9UOzOSASgjOILPzf8FA+lChjyztcxypJE12imvFiGEE
GBECACEFAkk5VooFCQlopH4HCwkIBwIDBAMWAQIFFQIBAwUChAEACgkQBsNOcLRJ
/JJ4WwCeJ8sgSXKNDEN4BsB4oC9jfZSAQocAn31RINEpxXlPA6MY1463j/U3oaSQ
=OTAo
-----END PGP PUBLIC KEY BLOCK-----
```

Public PGP key example. Source: https://docs.tibco.com/pub(mftis/8.3.0/doc/html/GUID-AB77E7DF-E9D8-4463-9582-19105FEE91A1.html

```
-----BEGIN PGP PRIVATE KEY BLOCK-----

lQOBBGC7U48RCACdbrfel9gzWMTvCDepXwpiCRlLyoUnJy0uCTpGPO+t0mQ1OUlD
Ahv1UpS6FSrqZOnMiM+QIOCTAJrzN+p6YFrvTcvQO9khsFzGgrGby29ZSTKzgOEa
Lr4aXv4IC0aON9splxb0340LDfrm261KL63g1cYP1RkWYw/Ch+dTGsq2Zjk21FEC
pS9kt9ZRMUPzMJJaCAQETkDJz+Mr3QEw+O5omb14C4HGUogCSatbv7gbpydp4SKf
J2B1zdNoTKKuQHcaScCHMElaNJbH1Du2Ri8/iiDeA0/U93JKbLpFmGZ/tsrBUxTv
dKk2v9jtu3QnYaBOX97Xf0HRpPQYXPFkd2bvAQCQFr3vNupu82PseR7kcIIdj7kd
JE9ZLiUujgD33MoknQf/SShh2UWK3Eaewq2ISHzMoZcGxUWHvK//ljZ/H9lnEI5t
sqxDfvAkDcmTRmbng4+D+V+k2GEP1a4ZtRa/KPSiZaEoP22NqS+uzNxoqcLgkRUd
```

Private PGP key example. Source: https://blogs.sap.com/2021/06/10/step-by-step-procedure-generate-pgp-keys-and-end-to-end-iflow-to-encrypt-and-decrypt-with-signatures

RSA (Rivest, Shamir, and Adleman, 1997)

- **The origin of public key systems**
  - The keys are based on two prime numbers (`p` and `q`)
  - The public key is a pair `(n, f(p,q))`, being `n = p*q`
  - The private key is a pair `(n, g(p,q))`
- **It is slow** (if compared with symmetric ones)
  - Data block encryption / decryption requires many operations with very large numbers
- **It is unfeasible to obtain the private from the public key**
  - **The only way to attack this system is to factor `n`**
    - Getting which two numbers have been multiplied to get `n`
    - Factoring **needs a lot of computation**
    - 2048-bit keys are considered computationally secure today
    - Many organizations are recommending migrating from 2048-bit RSA to 3072-bit+ RSA
      - Especially now, that there are researchers claiming that they broke RSA-2048 using a quantum computer that can be built with current technology
  - However, if possible, it is better to migrate to **elliptic curve cryptography**
    - This ensures better resistance to future quantum computer attacks and more efficiency

Elliptic-Curve Cryptography (ECC)

- **Public-key cryptography based on the algebraic structure of elliptic curves over finite fields**
  - This is a mathematical concept that will not be reviewed in this course
  - https://en.wikipedia.org/wiki/Elliptic-curve_cryptography
- **Elliptic curves are applicable for key agreement, digital signatures, pseudo-random number generators, and other tasks**
  - **E. g. Elliptic-Curve Diffie–Hellman (ECDH)**
    - Is a variant of the Diffie–Hellman key exchange protocol, but using elliptic-curve cryptography
  - They can also be used for encryption
    - By combining ECC key agreement with a symmetric encryption scheme
  - Different types of elliptic curves exist to be use with this type of ECC algorithms
    - Like the secp256r1 and X25519 curves (https://en.wikipedia.org/wiki/Curve25519) created to be used with ECDH

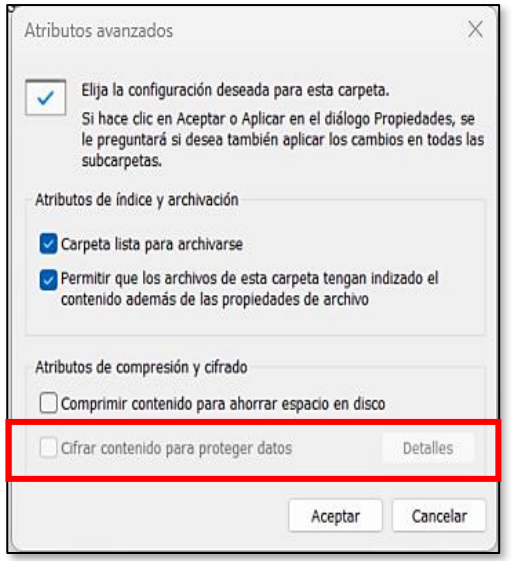
Elliptic-Curve Cryptography (ECC): Example

- The certificates (seen later) used to establish SSL/TLS connections use asymmetric cryptography
- There are two types
  - **Those that use RSA for their keys and those that use ECC cryptography**
  - ECC ones require **smaller keys** then non-ECC cryptography for attaining the same security level
  - This means that **ECC keys are faster and easier to handle** that equivalent RSA ones
    - And they scale better!
  - Therefore, **we should use ECC-based solutions**

ECC Key size	RSA equivalent key size
160 bits	1024 bits
224 bits	2048 bits
256 bits	3072 bits
384 bits	7680 bits
521 bits	15360 bits

Source: https://reanimandowebs.com/articulos-sobre-seguridad-web/certificado-ssl-tls/

Asymmetric algorithms: Practical applications



File and directory encryption

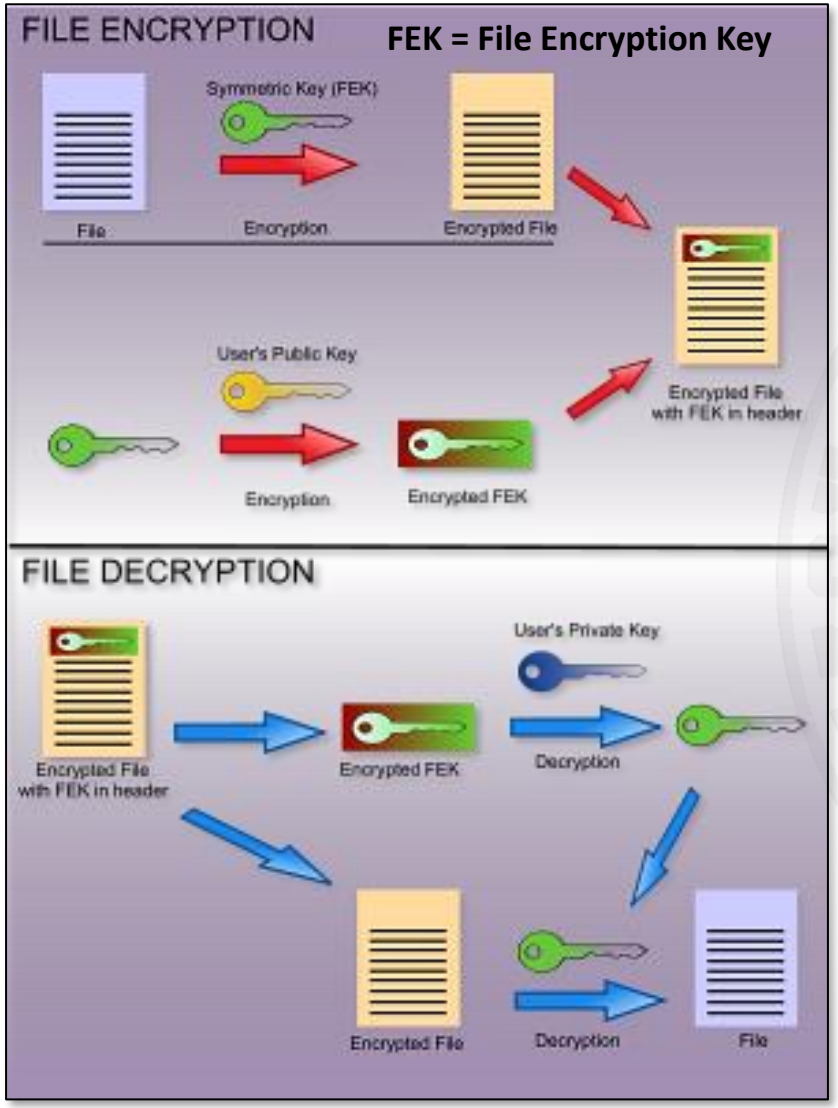
- **Individual files / directories are encrypted** by the file system
  - Often called file-based encryption, FBE, or file/folder encryption
  - Opposite to full disk encryption, where **the entire partition or disk is encrypted**
- **Types of filesystem-level encryption** include
  - '**Stackable**' cryptographic filesystems, layered on top of the main file system
  - General-purpose file systems with added encryption
- The **advantages** of filesystem-level encryption are
  - **Flexible file-based key management**: each file can be encrypted with a separate encryption key
  - **Individual management of encrypted files**: e.g., incremental backups of the individual changed files even in encrypted form, rather than backup of the entire encrypted volume
- **Combines public key cryptography and symmetric-key cryptography**
  - Access control can be enforced using public-key cryptography

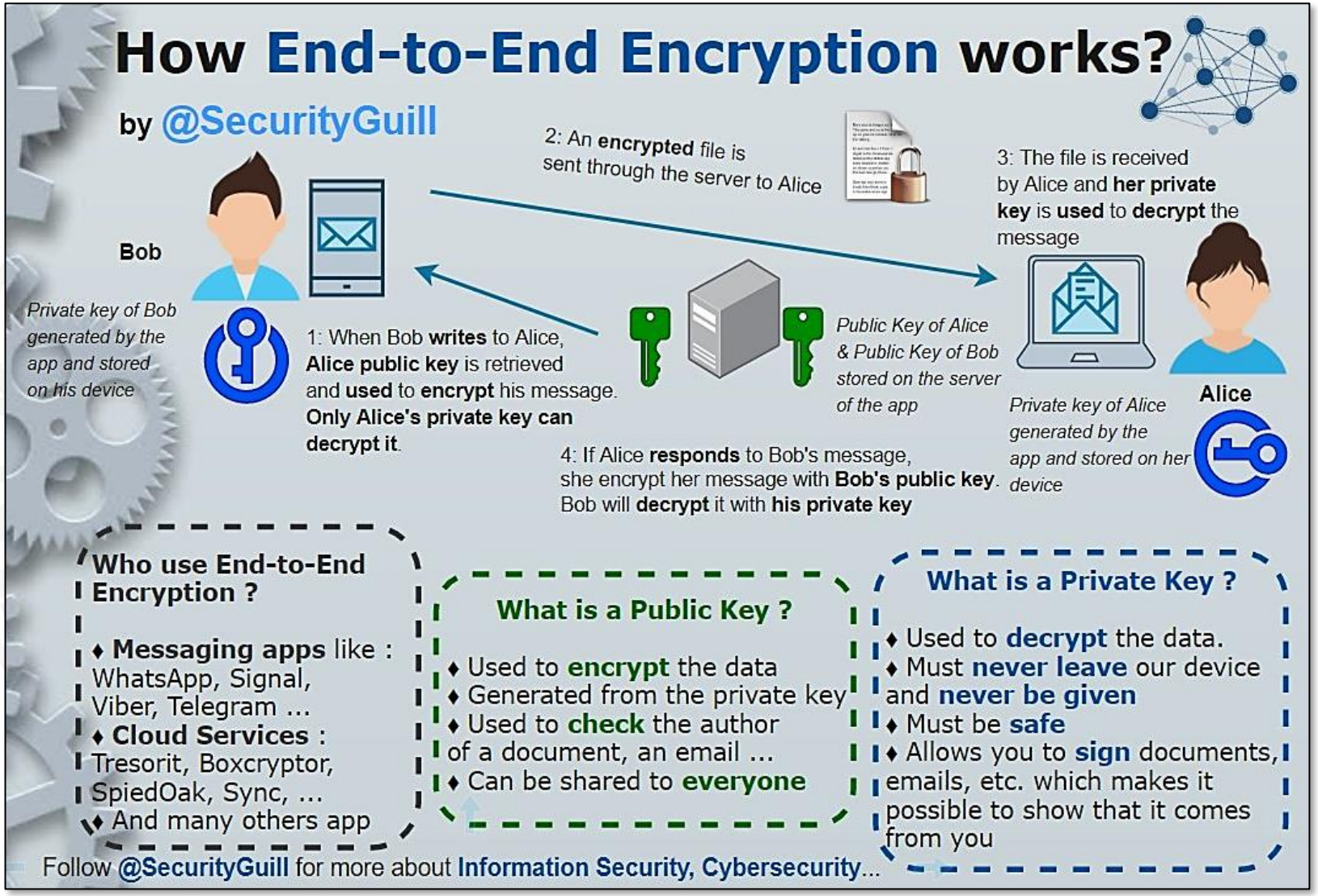
End-to-end encryption

Source: WhatsApp

- Secure communications between pairs of users, typical of lots of messaging and cloud applications

Asymmetric algorithms: Practical applications





Source: https://en.wikipedia.org/wiki/Encrypting_File_System

Used by WhatsApp, Telegram, Signal...

Think you've understood it all? Evaluate yourself!

?

- **You can do the following**
  - *Understand what block and stream encryption is*
  - *Understand the difference between keyed and non-keyed encryption*
  - *Understand what principles make a good cryptographic key and how you measure its strength*
  - ***Symmetric encryption***
    - *Be aware that, if you know the used cryptographic key, all the security of the encryption is lost and that, therefore, distributing the key is the biggest problem of the encryption*
    - *Know which symmetric algorithm you should use if you need to decide*
    - *Understanding what the Diffie-Hellman key exchange algorithm Is*
    - *Understand the concept and importance of Forward Secrecy in symmetric encryption*
    - *Be able to identify common applications of symmetric encryption in your daily life*
  - ***Asymmetric encryption***
    - *Clearly understand which key is distributed to others and which is not from the key pair*
    - *Understand the special properties of a pair of public/private keys, the relationship between them when encrypting/decrypting information, and that it is not possible to obtain one from the other*
    - *Understand how a typical cryptographic key looks like*
    - *Understanding why ECC encryption has an advantage over "traditional" encryption*
    - *Be able to identify common applications of asymmetric encryption in your daily life*

< Go to Index

INTEGRITY

Is this content unaltered?

*Icons by Bing Chat AI

Source: Bing Chat AI

Departamento de Informática
Universidad de Oviedo

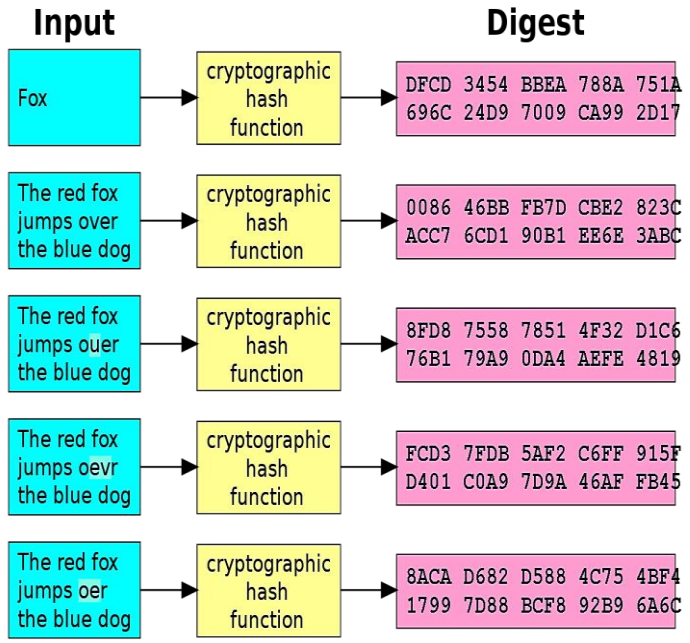
Universidad de Oviedo

What are you going to find in this block?

- **In this block you will learn**
  - What is a **hash function**, its main properties, and what is it used for
  - The **main current applications** of a hash function
  - What a **KDF** is and what it is used for, as well as the importance of what they do

HASH OR SUMMARY FUNCTION

- A mathematical function that transforms a set of data M (text, file…) into a **fixed-length** (n) output H
  - Therefore, an arbitrary-length input message M is represented by the **fixed-length** (n) output of the Hash(M) function, named **H**
  - The output (H) is called **cryptographic summary, fingerprint, or digest**
- Properties
  - **Easy to calculate**: it is computationally cheap to obtain H=Hash(M)
  - **Unidirectional:** It is impossible to get the original message M from H
  - **Compression**, **diffusion** and **non-predictability**
    - Favors the "avalanche effect"
    - Fixed-length output implies **that several different messages M may generate the same value H=Hash(M)**
    - These 3 properties ensure that it is **very difficult to predict** which messages have the same hash
    - Also, that changing even just one bit on M generates a very different H



Input message size varies, but the hash size Is always the same!

Source:
https://es.wikipedia.org/wiki/Funci%C3%B3n_hash_criptogr%C3%A1fica

HASH OR SUMMARY FUNCTION

- **First preimage resistance**
  - If you know a hash value H, finding or creating a message that generates it must be very difficult
  - Depends on the size n of the H calculated by the hashing algorithm we use
- **Second preimage resistance** (simple and strong collision resistance)
  - **Simple**: It is computationally difficult that given a M, we can find a M' so that Hash(M) = Hash(M')
    - Making variations of M must not generate the same hash (avalanche effect, prevents message forgery)
  - **Strong**: It is computationally difficult to find a random pair of messages (M, M') that Hash(M) = Hash(M')
    - Random messages generating the same hash must not be easily found (forgery)

- **There are many hashing algorithms**
  - **MD5** (Message Digest Algorithm, 128 bit) and **SHA-1** (Secure Hash Algorithm, 160-bit): **Insecure**
  - **SHA-2** algorithms **are the recommended ones** nowadays
    - **SHA-256**, **SHA-384**, or **SHA-512**
  - **SHA-3.** Current NIST standard that could be more future-proof
    - Hash length varies: 224, 256, .., 512 bits (http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf)
  - **BLAKE2** is another future-proof hash algorithm

Hash algorithms applications

- **Password storage**: increase the difficulty of knowing user passwords
  - Usually forcing attackers to rely on computationally expensive brute-force techniques
  - Prevents easily obtaining passwords if a password database is stolen (common attack)
  - Storing passwords in plain text is a clear sign of a company **not caring about security at all**
  - Brute-force may be not feasible **if the password is complex enough**
  - A KDF (Key Derivation Function) is used to ensure maximum resistance against attacks (seen later)

```
gates:$1$vjFpMmig$2yJJhqiYADW0w03tPQZB9.:50:0:99999:7: ::
elon:$1$0QT3fPmk$UHSi7AcfaLvXP49P3gbvb/:100:0:99999:7:::
```

- **File fingerprinting**: Ensuring that a file is the original one
  - Check that no modifications happened during the transmission / storage (trojanized, corrupted…)
  - E. g. a program file can also publish its computed hash using a certain hashing algorithm
    - Compute the executable hash locally, and know if the file has been altered or corrupted
    - Just modifying a single bit outputs a different hash result
- **Digital signatures:** ensure that a file was created by who claims it (seen later)
- **Blockchain**: Uses the SHA-256 algorithm to detect fraudulent chain modifications

File fingerprinting: Checking file hashes

- **Hashes may be used to detect unauthorized modifications to our software**
- **Host-Based Intrusion Detection Systems (HIDS, like Tripwire or AIDE) precalculate and securely store the hashes of critical system files**
  - They also periodically check these hashes
  - Hash mismatch between stored and calculated ones may mean
    - **A new version of the executable is present**: software update; stored hash must be recalculated
    - **Malicious file alteration**: malware is in the system
- **Executables may also check its own hash upon launch, but this is less secure**
  - Code that perform the check may be disabled by malware
  - The correct hash must be stored / obtained from a remote site, and it may be tampered with
  - **Microsoft SmartScreen uses this approach**: the OS analyzes the file, evaluates its "reputation", and send a hash to a cloud service to decide if it will be run (or warns the user about potential dangers)

```
6de6036ef0dc8a908e4cc248ef1d8aab87172e722d8c5bad9e137fd43994e0fe  ./netboot/debian-installer/amd64/grub/x86_64-efi/parttool.lst
13886a4e494c33bcf387210f6c6910bc4a84db81d931b63ecc2f61ad722fb483  ./netboot/debian-installer/amd64/grub/x86_64-efi/parttool.mod
7aa3c84b739ed7920e21de6ef6013c50e797adb4aa5d86e92cceeac3536bb0e5  ./netboot/debian-installer/amd64/grub/x86_64-efi/password.mod
3c0ff52b79422033b3aa3153be7f67473dcec3e34155c1c725daba8abb96a6ec  ./netboot/debian-installer/amd64/grub/x86_64-efi/password_pbkdf2.mod
dc9bdda70583365f2cc7e63417d1e056f876067cb3bf67fa43c5ae51f112ea36  ./netboot/debian-installer/amd64/grub/x86_64-efi/pata.mod
dd91bce28a6e9b7c801e38ed2a8ab9ce1aed8970b9880054c22f438a16f9d30f  ./netboot/debian-installer/amd64/grub/x86_64-efi/pbkdf2.mod
5a6300a6b576afa0a96685571c0af13bd69041b248a7494db825e083dfa0106c  ./netboot/debian-installer/amd64/grub/x86_64-efi/pbkdf2_test.mod
50eec78eb440928415493c0c59cf11ce2d909c582b3beea58c590894f437fd32  ./netboot/debian-installer/amd64/grub/x86_64-efi/pcidump.mod
```

Source: https://deb.debian.org/debian/dists/bullseye/main/installer-amd64/current/images/SHA256SUMS

Password storage: Key Derivation Function (KDF)

- **Typical techniques to guess plain-text passwords from stored data**
  - **Pure brute-force**: covering all the possible key space and normally with a prohibitive cost
  - **Dictionaries** (EDI) of potential keys that may correspond to stored data
    - We check less values, but consume more memory and may be unsuccessful (key not present in the dictionaries we used)
  - **Rainbow tables**: data structures that allow to store large password quantities
    - Consuming less memory but requiring more computational power that plain dictionaries
- **Can we resist them?**
  - Yes, but not using plain hash algorithms
  - But specialized Key Derivation Functions (KDF) ones!

➢ Exercises **L3B3_SSLCRACK**, **L3B3_PASS**, **L3B3_BRUTE**

Password storage: Key Derivation Function (KDF)

- **KDF obtains one or more keys from a secret value**
  - The secret value is the **master key**, the actual password that the user inputs ☺
  - Their output is used to store passwords
    - Making password-guessing techniques more expensive
  - How? using "**key stretching**" techniques to enhance the security of the stored data
    - **Salt** (or seed): Random data concatenated to the plain text passwords
      - Used to make dictionary or rainbow-table-based attacks more difficult
      - Salts ensure that the same password does not always generate the same hash
      - Unique for each password, and normally stored along a password hash
      - A pepper (or secret salt) is also a secret added to an input during hashing
        - But they are stored separately in an external medium, such as a Hardware Security Module
    - **Iterations/rounds**: apply the hash function several rounds (>=1)
      - Obtaining the original text with brute force is more expensive, delaying each attempt

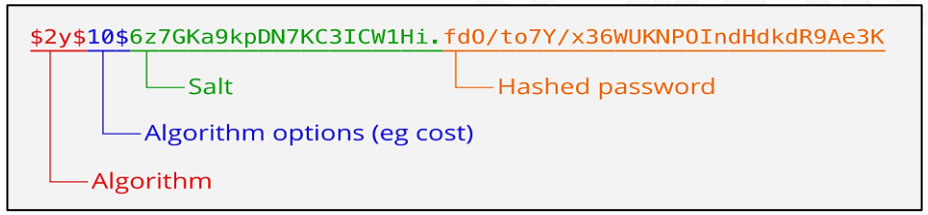
```
DK (Derivation Key: data to be really stored) = KDF (user-supplied password or key, Salt, Iterations)
```

Password storage: Key Derivation Function (KDF)

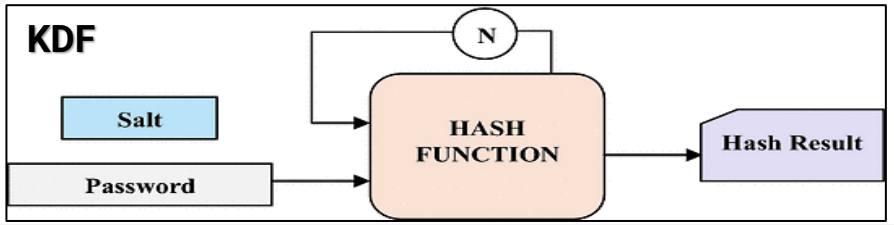
- **Resistance to these attack types depend on the key length and the amount of computational entropy that could be generated while generating the DK**
  - The more, the better security
  - This is the randomness collected by an OS / application to use in cryptography, among other uses
- **Known modern KDF algorithms**
  - **PBKDF2**: 160-bit length, uses salt, and recommended iterations from 1000 to 10000000
    - Used by Wifi networks, Truecrypt, Veracrypt, the GRUB bootloader, Android…
  - **scrypt**: forces using a lot of memory, so implementing attacks against their values costs more
    - Used as a proof-of-work in some cryptocurrencies
  - **bcrypt**: The password hashing algorithm of several Linux distros, uses salt and iterations
  - **Argon2id**: It has three versions specifically parametrized against different attacks

Source: https://stackoverflow.com/questions/40993645/understanding-bcrypt-salt-as-used-by-php-password-hash

Bcrypt stored string example



Source: https://en.bitcoinwiki.org/wiki/Key_derivation_function



Password storage: Key Derivation Function (KDF) practical example

- **This image shows the content of the `/etc/shadow` file**
  - Linux passwords are stored there
  - `*` means that the user does not have an interactive (bash) account: it is a **service account**
  - The users `vagrant`, `redondo`, and `vinuesa` have the same password! ("`test123…`")
- **Why their hash is so different?**
  - **Remember**: pepper and salt; same password does not generate the same hash
    - If the `vagrant` hash is stolen, they will not know `vinuesa` password automatically, even if it is the same!
    - If `/etc/shadow` is stolen and a hash is broken, they will not immediately know the passwords of other users that may use the same password
    - Do you understand the importance of storing passwords like this to minimize the effects of a data leak?
    - And `redondo`? It is using a weaker hashing algorithm (and shorter), but following the same principles

```
vagrant:$6$zGFNTxdh$RZWHA/hxHzu/CNAnZyJxBiKGZB0UpZwU0OpwDSpI/vOOGm8hgArmO1FC9/xLV6pe6xLvuGB5EJ.WyzguLukDY1:18983:0:99999:7:::
vboxadd:!:18123::::::
rtkit:*:18983:0:99999:7:::
usbmux:*:18983:0:99999:7:::
pulse:*:18983:0:99999:7:::
redondo:$1$c26Kbia1$d/7sXVp9im42R/w.oqU2U/:18983:0:99999:7:::
vinuesa:$6$Ikh3d1YG$RyOzFnaoxtqAZMqLyy3a0ftVI.B5EkUEHCeGSuJxNM2u5FIy46XYQ1IeeLR4BTlwhydw11sfnwwRUjqVsPM.5/:18988:0:99999:7:::
```

Cracking a password could be impossible if it is strong enough…

Source:
https://www.reddit.com/r/dataisbeautiful/comments/322lbk/time_required_to_bruteforce_crack_a_password/

Length of password approximately equivalent to a given entropy

Note: First three columns are rounded to a max of +/- 0.2
Fourth column is rounded to a max of +/- 0.05

Time until *guaranteed* brute-force password crack

Based on attacker's guesses per second vs. password strength

Formula: (*Seconds to guaranteed crack*) = ($2^{(entropy)}$) ÷ (guesses per second)

Result then converted from *seconds* to more reasonable units of time such as years

Note: By definition, it takes **half** of the *guaranteed* crack time on *average* to crack a password

Entropy (in bits)
Entropy = $\log_2(S^L)$
L is pass length
S is symbol pool
(*i.e. if your password has lowercase and numbers, your **symbol pool** is 26+10=36*)
(*If entropy confuses you, just focus on the rest of the chart*)

Entropy	Lowercase (26 symbols, 4.7 bits ea)	UPPER + lower + 0-9 (62 symbols, 5.95 bits ea)	UPPER + lower + 0-9 + special characters (94 symbols, 6.55 bits ea)	Pass **phrase** with an average of **4.3** letters per word (12.93 bits per word) Column's value = words in phrase	Attacker's brute force guesses per second: 10,000	1,000,000	1,000,000,000	100,000,000,000	1,000,000,000,000	100,000,000,000,000	Entropy
120		20			∞	∞	∞	421 quadrillion years	42 quadrillion years	421 trillion years	**120**
118	25		18		∞	∞	∞	105 quadrillion years	11 quadrillion years	105 trillion years	118
116				9	∞	∞	∞	26 quadrillion years	2.6 quadrillion years	26 trillion years	116
114		19			∞	∞	658 quadrillion years	6.6 quadrillion years	658 trillion years	6.6 trillion years	114
112	24		17		∞	∞	165 quadrillion years	1.6 quadrillion years	165 trillion years	1.6 trillion years	112
110					∞	∞	41 quadrillion years	411 trillion years	41 trillion years	411 billion years	**110**
108	23	18			∞	∞	10 quadrillion years	103 trillion years	10.3 trillion years	103 billion years	108
106			16		∞	∞	2.6 quadrillion years	26 trillion years	2.6 trillion years	26 billion years	106
104	22			8	∞	643 quadrillion years	643 trillion years	6.4 trillion years	643 billion years	6.4 billion years	104
102		17			∞	161 quadrillion years	161 trillion years	1.6 trillion years	161 billion years	1.6 billion years	102
100					∞	40 quadrillion years	40 trillion years	402 billion years	40 billion years	402 million years	**100**
98	21		15		∞	10 quadrillion years	10 trillion years	100 billion years	10 billion years	100 million years	98
96		16			251 quadrillion years	2.5 quadrillion years	2.5 trillion years	25 billion years	2.5 billion years	25 million years	96
94	20				63 quadrillion years	628 trillion years	628 billion years	6.3 billion years	628 million years	6.3 million years	94
92			14		16 quadrillion years	157 trillion years	157 billion years	1.6 billion years	157 million years	1.6 million years	92
90	19	15		7	3.9 quadrillion years	39 trillion years	39 billion years	392 million years	39 million years	392 millennia	**90**
88					981 trillion years	9.8 trillion years	9.8 billion years	98 million years	9.8 million years	98 millennia	88
86			13		245 trillion years	2.5 trillion years	2.5 billion years	24.5 million years	2.5 million years	25 millennia	86
84	18	14			61 trillion years	613 billion years	613 million years	6.1 million years	613 millennia	6.1 millennia	84
82					15.3 trillion years	153 billion years	153 million years	1.5 million years	153 millennia	1.5 millennia	82
80	17				3.8 trillion years	38 billion years	38 million years	383 millennia	38 millennia	383 years	**80**
78		13	12	6	958 billion years	9.6 billion years	9.6 million years	96 millennia	9.6 millennia	96 years	78
76	16				239 billion years	2.4 billion years	2.4 million years	24 millennia	2.4 millennia	24 years	76
74					60 billion years	599 million years	599 millennia	6 millennia	599 years	6 years	74
72		12	11		15 billion years	150 million years	150 millennia	1.5 millennia	150 years	1.5 years	72
70	15				3.7 billion years	37 million years	37 millennia	374 years	37 years	4.5 months	**70**
68					935 million years	9.4 million years	9.4 millennia	94 years	9 years	34 days	68
66	14	11	10		234 million years	2.3 million years	2.3 millennia	23 years	2.3 years	8.5 days	66
64				5	58 million years	584 millennia	584 years	5.8 years	7 months	2.1 days	64
62	13				14.6 million years	146 millennia	146 years	1.5 years	1.8 months	13 hours	62
60		10	9		3.7 million years	37 millennia	37 years	4.4 months	13 days	3.2 hours	**60**
58					913 millennia	9.1 millennia	9 years	33 days	3.3 days	48 minutes	58
56	12				228 millennia	2.3 millennia	2.3 years	8.3 days	20 hours	12 minutes	56
54		9			57 millennia	571 years	6.8 months	2.1 days	5 hours	3 minutes	54
52	11		8	4	14.3 millennia	143 years	1.7 months	12.5 hours	1.3 hours	45 seconds	52
50					3.6 millennia	36 years	13 days	3.1 hours	19 minutes	11 seconds	**50**
48	10	8			892 years	8.9 years	3.3 days	47 minutes	4.7 minutes	2.8 seconds	48
46			7		223 years	2.2 years	20 hours	12 minutes	1.2 minutes	0.7 seconds	46
44					56 years	6.7 months	4.9 hours	2.9 minutes	18 seconds	0.18 seconds	44
42	9	7			14 years	51 days	1.2 hours	44 seconds	4.4 seconds	0.044 seconds	42
40			6	3.1	3.5 years	13 days	18 minutes	11 seconds	1.1 seconds	0.011 seconds	**40**

Entropy (in bits)
Entropy = $\log_2(S^L)$
L is pass length
S is symbol pool
(*i.e. if your password has lowercase and numbers, your **symbol pool** is 26+10=36*)
(*If entropy confuses you, just focus on the rest of the chart*)

Lowercase (26 symbols, 4.7 bits ea)
UPPER + lower + 0-9 (62 symbols, 5.95 bits ea)
UPPER + lower + 0-9 + special characters (94 symbols, 6.55 bits ea)
Pass **phrase** with an average of **4.3** letters per word (12.93 bits per word) Column's value = words in phrase

10,000 | 1,000,000 | 1,000,000,000 | 100,000,000,000 | 1,000,000,000,000 | 100,000,000,000,000

Attacker's brute force guesses per second

Time until *guaranteed* brute-force password crack

Based on attacker's guesses per second vs. password strength

Formula: (*Seconds to guaranteed crack*) = ($2^{(entropy)}$) ÷ (guesses per second)

Result then converted from *seconds* to more reasonable units of time such as years

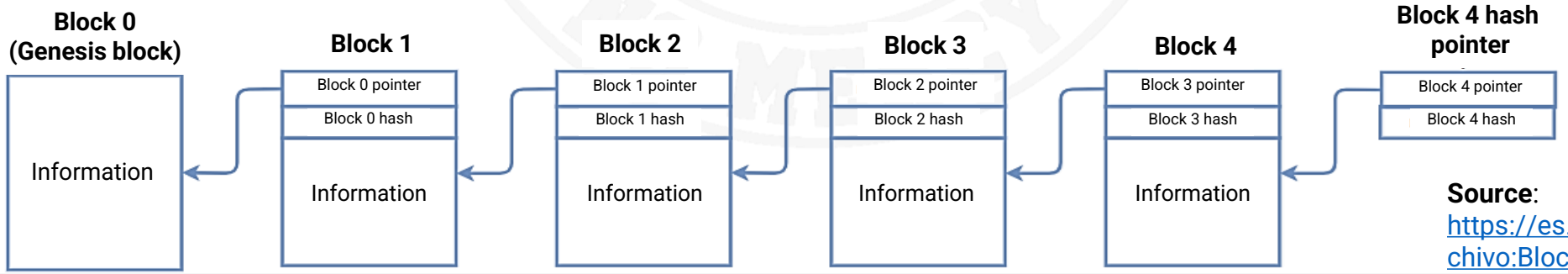
Note: By definition, it takes **half** of the *guaranteed* crack time on *average* to crack a password

Length of password approximately equivalent to a given entropy

Note: First three columns are rounded to a max of +/- 0.2
Fourth column is rounded to a max of +/- 0.05

BLOCKCHAIN

- **The first block (block 0) of the chain is known as the genesis block**
  - Each other block stores a pointer to the previous one, its hash value, and their information
- **It is impossible to change one block of the chain without being detected by the others**
  - If an attacker changes the contents of block 1, due to collision resistance it will not be feasible to find other information that has the same hash as the previous info: the hash value of block 1 changes
  - **Block 2 will now be able to detect this change on block 1**, since the hash it saves will not match
  - To prevent this, the attacker would have to change the value of block 1 hash on block 2
  - But that will modify the hash of block 2, so now block 3 would be able to detect the problem
  - This situation is repeated throughout the chain, until the end
  - Users save the final block hash, so if you try to modify it, they will detect the inconsistency again
- **That's why blockchain is an ideal tool for storing non-counterfeit transactions**

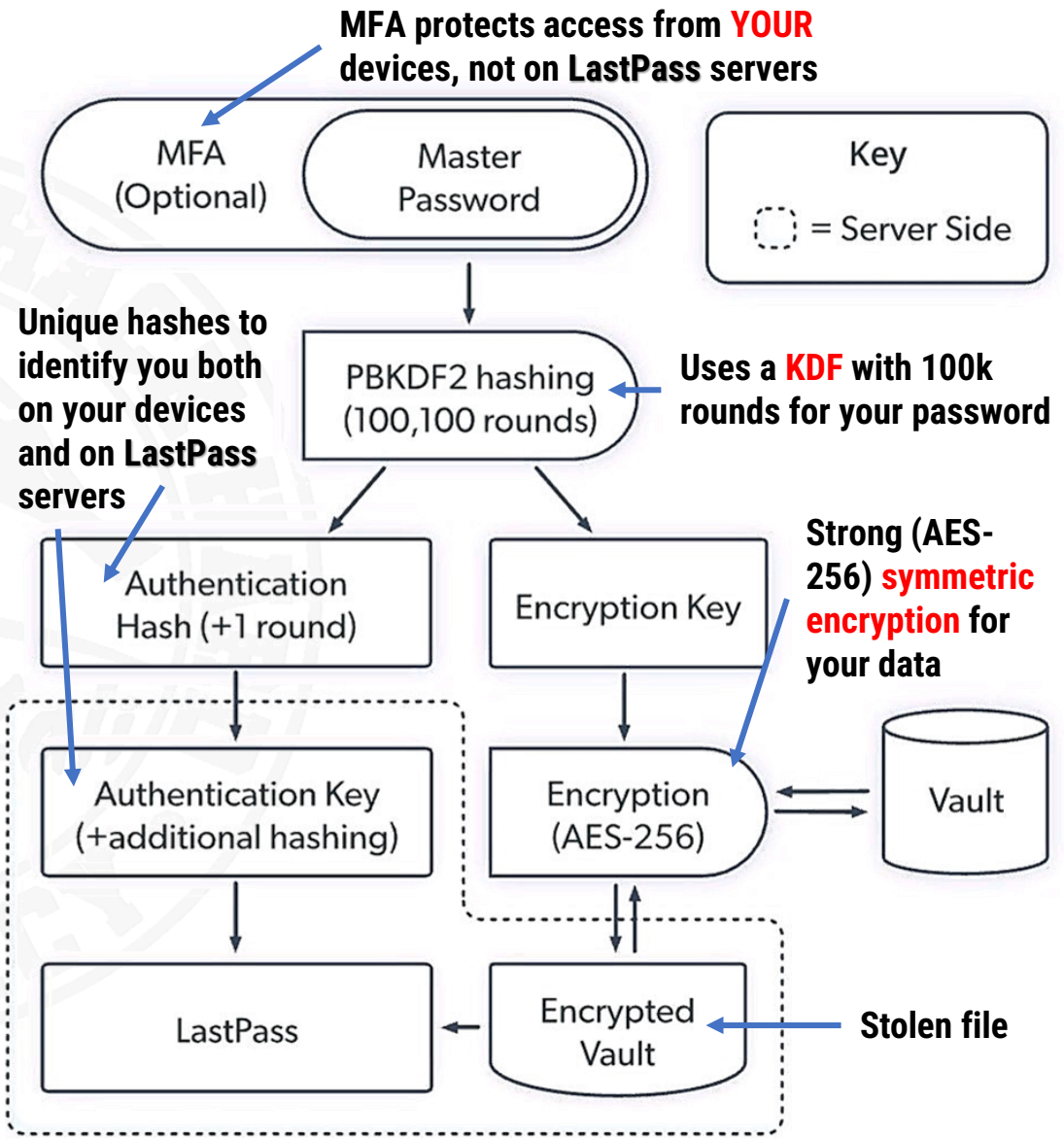


Source: https://es.wikipedia.org/wiki/Archivo:Blockchain_Hash.png

Practical example: The 2022 LastPass leak

Seguridad de Sistemas Informáticos

- In 2022 LastPass reported that all its user encrypted vaults were copied from its servers
- This potentially compromises all users!
  - A weak master password **can be brute-forced offline** by the attackers
    - https://www.tomshardware.com/news/eight-rtx-4090s-can-break-passwords-in-under-an-hour (**288.500.000.000 hashes/s!**)
  - All the user's passwords would be known!
  - **Users need to reset ALL its passwords** just in case...
- LastPass used encryption correctly…
  - KDF with high number of rounds, AES-256, no storage of user's master passwords server-side, hashing...
- **But failed on the authentication side!**
  - Somebody could access other user's files
  - Even encrypted, this is sensitive information!



Source: https://www.linkedin.com/feed/update/urn:li:activity:7012507887449116672/

Think you've understood it all? Evaluate yourself!

?

- **You can do the following**
  - *Be aware that the size of a hash only depends on the algorithm you use to create it, and not on the size of the input*
  - *Clearly understand the main properties of a hash, especially its unidirectionality and that, with equal input, equal hash*
  - *Understand that hashes have a series of properties that cause them to change a lot just by changing one bit of the input*
  - *Understand why hashes are ideal for securely storing passwords and verifying that a file has not been modified*
  - *Understand what a KDF is and what it's used for, as well as the operations that make it the ideal form of hashing to store keys*
  - *Understand the different methods used to crack passwords and protection against them*
  - *Understand the basic principles of how a blockchain works and how it uses hashes to do so*

< Go to Index

Source: Bing Chat AI

Digital sign >

Auth. Encryption >

Certificates >

AUTHENTICATION

Who do I think you are?

*Icons by Bing Chat AI

What are you going to find in this block?

- **In this block you will learn**
  - What a **digital signature** exactly is
    - And the traditional **procedure** of digitally signing a document
  - What **authenticated encryption** is used for and what problem it solves
  - What exactly is **a digital certificate** and what is it used for
  - How to resolve the **trust issue** between a certificate and who says it owns
  - The **main uses** of digital certificates

Digital Signatures

Not the photo of your physical signature! ☺

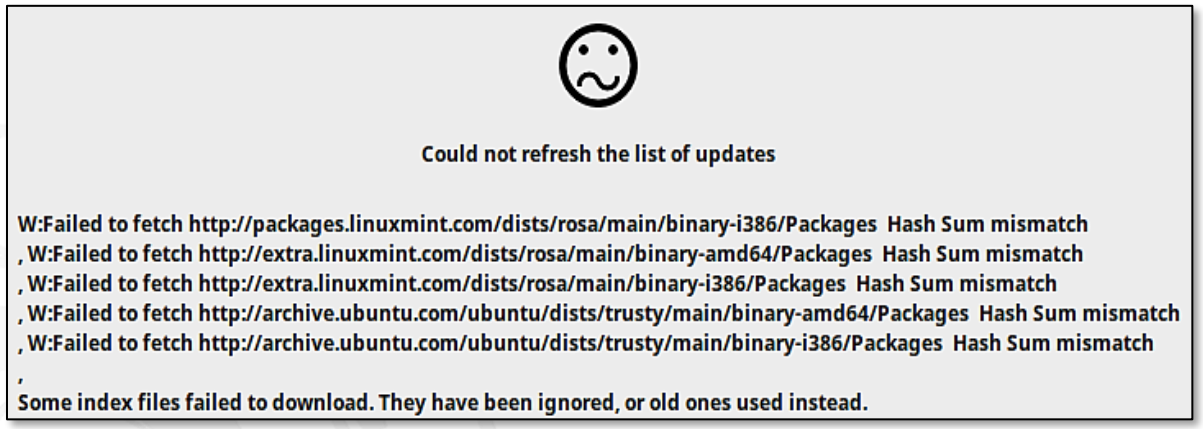
*Icons by Bing Chat AI

DIGITAL SIGNATURE

- Result of a cryptographic operation that joins a signed document to a personal element, the **Digital Certificate**
- They are the result of **cryptographically signing the hash of a file** with its author's private key
  - Signatures are usually files with a `.sig` or `.asc` extension
- A signed file can be verified
  - The original file hash can be obtained (along with the user's public key) from the signed file
  - If the document was modified in any way, this hash will be different from the hash that will be calculated from the current contents of the file
  - The attempt to verify the signature would fail, and **file tampering will be detected**
- Therefore, they certify a document and adds a timestamp to it
- An example implementation is the **Digital Signature Standard** (DSS):
  - https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-5.pdf

Digital signature. Objectives

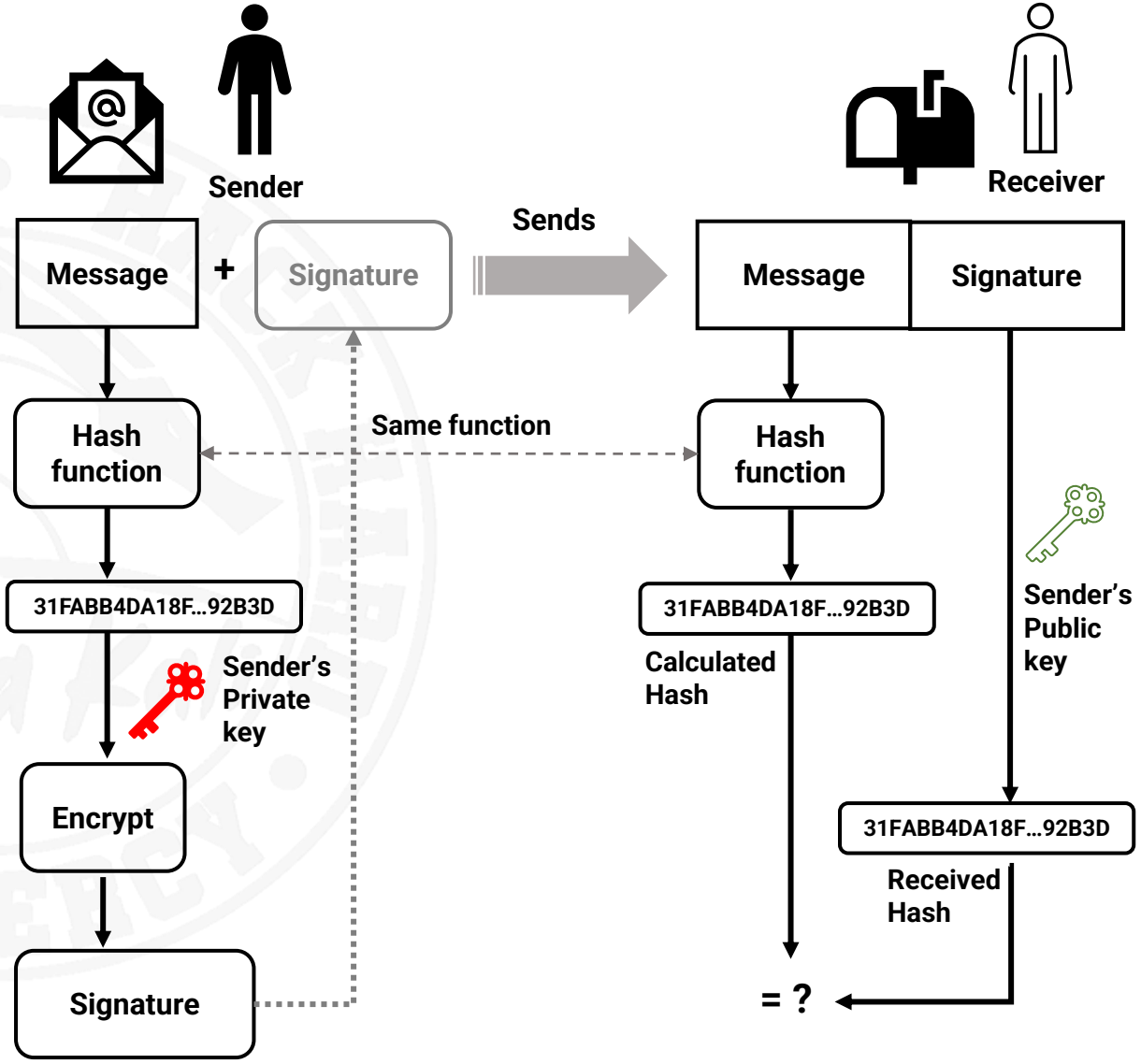
- **Authentication of origin**
  - The signature must convince the receiver that **the sender has deliberately signed the document**
- **Integrity**
  - Must ensure that **the document / message has not been modified**
- **Do not repudiate**
  - By guaranteeing the previous two, **the author cannot deny the content of the message**
- **It must be unfalsifiable, not be reusable and, if the document is altered, it must be possible to detect it**



Source: https://www.comoinstalarlinux.com/como-solucionar-problemas-al-instalar-o-actualizar-paquetes-en-ubuntu-o-linux-mint/

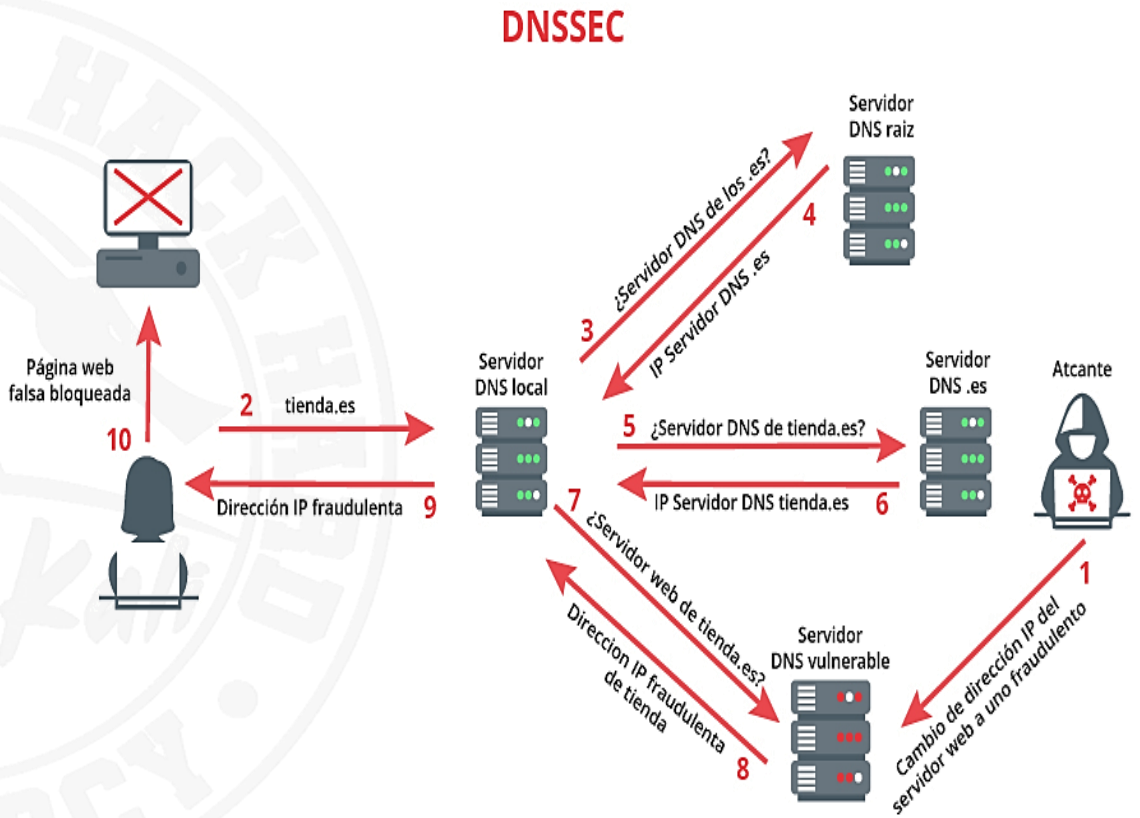
Digital signature & Hash functions: Message signing procedure

- **A message hash is calculated and ciphered with user's private key**
  - The computational cost of ciphering a hash is much lower than ciphering the full message
    - Hash size is fixed, smaller than the message, and represents it uniquely
    - Ciphering the hash is equivalent to ciphering its corresponding message
- **Hash is deciphered once received**
  - With the user's **public key**
  - The result can be checked against the calculated hash of the received message
- **If they are equal, the message has been unaltered**
  - And we know who the sender is!



APPLICATIONS OF DIGITAL SIGNATURES: DNSSEC

- Topic 1 described the DNS resolution "chain"
- But if one of those DNS servers is attacked and its data compromised...
  - The IP of any domain name on Internet could be falsified!
  - **Internet would not be trustable**, and fall apart ☹
- DNSSEC uses digital signatures to ensure that the contents (DNS addresses) of the servers are not tampered with
  - It uses a "chain of custody" that clients can verify
  - Any unauthorized domain name modification will be detected and rejected
- More info
  - https://www.infoblox.com/dns-security-resource-center/dns-security-solutions/dns-security-solutions-dns-security-extensions-dnssec/



Source: https://www.incibe.es/protege-tu-empresa/blog/dnssec-asegurando-integridad-y-autenticidad-tu-dominio-web

When digital signatures fail: The Sony PS3® case

- **Legal games in the PS3® console must be signed by a Sony private key to prevent data corruption and ensure authentic game execution**
  - Consoles are shipped with the corresponding public key in its firmware
  - The console will validate the hash of the game files with its public key, refusing to execute software that do not pass the validation
- **But this private key was stolen and published on Internet in 2011**
  - And therefore, anybody can sign software with it...
  - ...so people could create executables able to run in the PS3® hardware (homebrew)
  - Sony Mario Bros could be a possibility! ☺
- **The PS3® scene was then a race between modders and Sony to prevent unauthorized software execution**
  - By developing new firmware and PSN network checks
- **Conclusion: A private key leak caused millions of losses to Sony**

Authenticated Encryption

MACs without Apples ☺

*Icons by Bing Chat AI

Why is this type of encryption used?

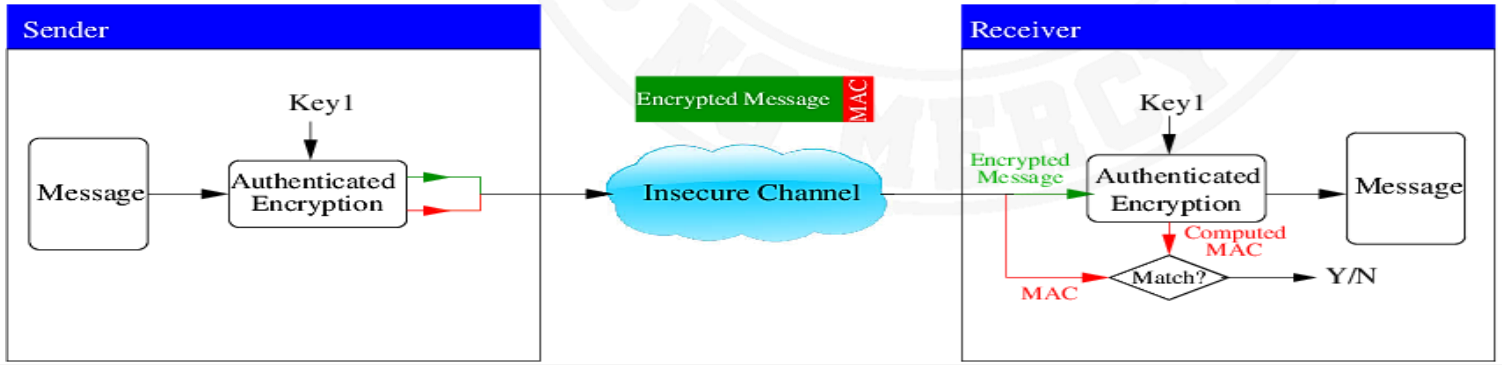
- **We saw that to guarantee CIA we need to use asymmetric encryption**
  - But it is slow, specially when transmitting big volumes of data
- **Also, that symmetric encryption is faster, but only guarantees confidentiality**
  - Ciphered data can be corrupted, and you also have the key distribution problem
- **To achieve CIA efficiently, both ciphering types were combined in the past**
  - But turned to be error-prone historically, because it was complex to combine them correctly
- **For that reason, a new cipher type was created: Authenticated Encryption (AE)**
  - **Symmetric ciphering** able to detect if the message has been altered (integrity)
  - Uses **a unique shared key** by a concrete sender (S) and receiver (R), so they are authenticated
    - Only R can know what S sends and vice versa
  - *What if we use asymmetric encryption to only share a random symmetric shared key securely, and later AE for the rest of the traffic using that shared key?*
  - Best of **both worlds**! Fast encryption with no key distribution problem ☺

AUTHENTICATED ENCRYPTION

- So, many modern security protocols (TLS, WPA2, WireGuard, Signal…) use AE
  - That optionally can carry Authenticated Data (AEAD)
    - The Authenticated Data is not encrypted, so it can be read by the receiver
    - But if it is tampered with, **the algorithm will detect it**
- AEAD is a variant of symmetric encryption that additionally provides authentication
  - Therefore, **both sender and receiver must know a shared symmetric key**
  - For example, in TLS, ciphertext is constructed with an AEAD
    - Composed by a plaintext header (AD) and a ciphered payload (AE)
    - Both are authenticated, but **only the payload is encrypted**
    - The plaintext header includes the content type and the ciphertext length, needed to decipher the payload once received
- More information
  - https://textbook.cs161.org/crypto/macs.html
  - https://www.cryptography-primer.info/encryption/

MAC (Message Authentication Code)

- **Using AE o AEAD adds to the message a Message Authentication Code or MAC**
  - A MAC is a ciphered **checksum** of the message, sent along it
    - **A checksum is a hash type,** but specialized in message integrity checks
    - https://www.baeldung.com/cs/hash-code-vs-checksum
  - The MAC is ciphered using the shared symmetric key
  - The resulting MAC has fixed (and small) length
  - A MAC ensures that any change to the message **makes the checksum invalid**
- **There are 3 ways of creating MACs**
  - **Hash-based** (HMAC, **recommended**), **Block cipher-based** (CBC-MAC), and **Galois Counter Message Authentication Code-based** (GMAC)

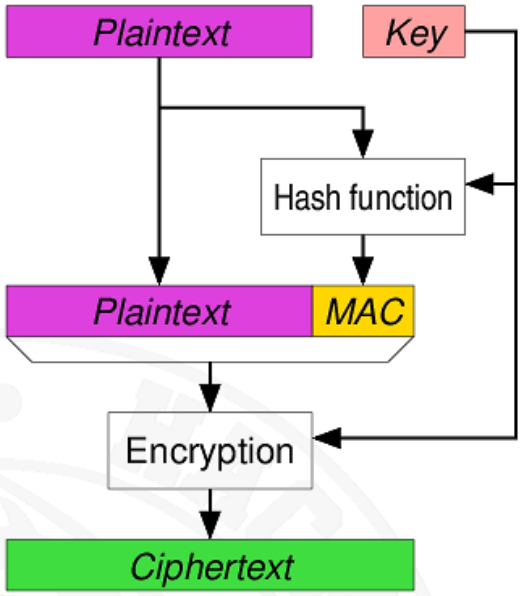


Source: https://www.researchgate.net/publication/279264224_Authenticated_Encryption_on_FPGAs_from_the_Reconfigurable_Part_to_the_Static_Part/figures?lo=1

MAC CALCULATION

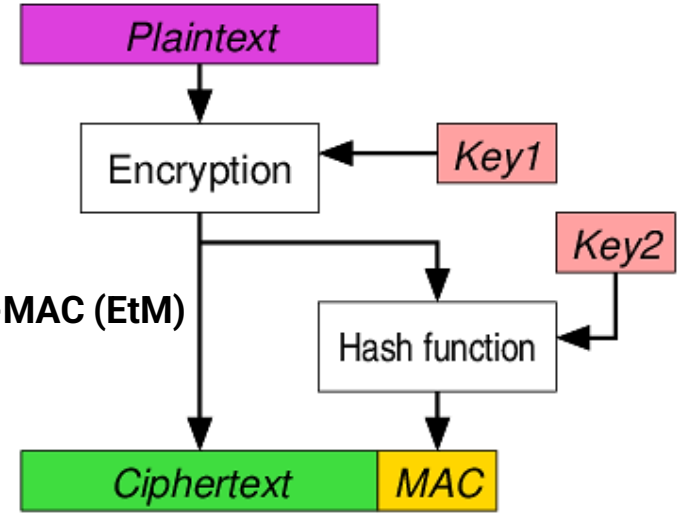
Seguridad de Sistemas Informáticos

- **We start with 2 users S and R sharing a secret key K**
  - S creates a MAC from the message M with a function f
    - MAC = f(K, M)
    - It is impossible to obtain K from MAC and M, **f is not reversible**
  - S sends M+MAC to R
  - R calculates MAC' with K and M
  - If MAC' = MAC, the message is authenticated
- **There are three ways of doing this**
  - Mac-then-Encrypt (MtE): used by SSL
  - Encrypt-then-MAC (EtM): used by IPSec, the recommended method
  - Encrypt-and-MAC (E&M): used by SSH

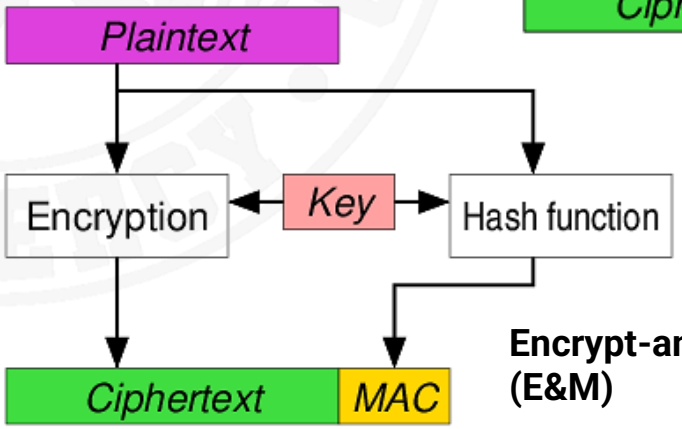


Source: https://en.wikipedia.org/wiki/Authenticated_encryption

Mac-then-Encrypt (MtE)



Encrypt-then-MAC (EtM)



Encrypt-and-MAC (E&M)

Think you've understood it all? Evaluate yourself!

?

- You can do the following
  - ***Digital Signatures***
    - *Have a good understanding of how hashes and asymmetric encryption are used to sign a document*
    - *What a signature of this kind guarantees*
    - *Understand how signing ensures trust in the operation of a critical system, such as a DNS or software authentication system on closed hardware*
    - *As a result, understanding that a leak of a private key used in a signing compromises the entire signature system*
  - ***Authenticated encryption***
    - *Understand the problem that authenticated encryption solves over typical symmetric encryption*
    - *Understanding what a MAC is*

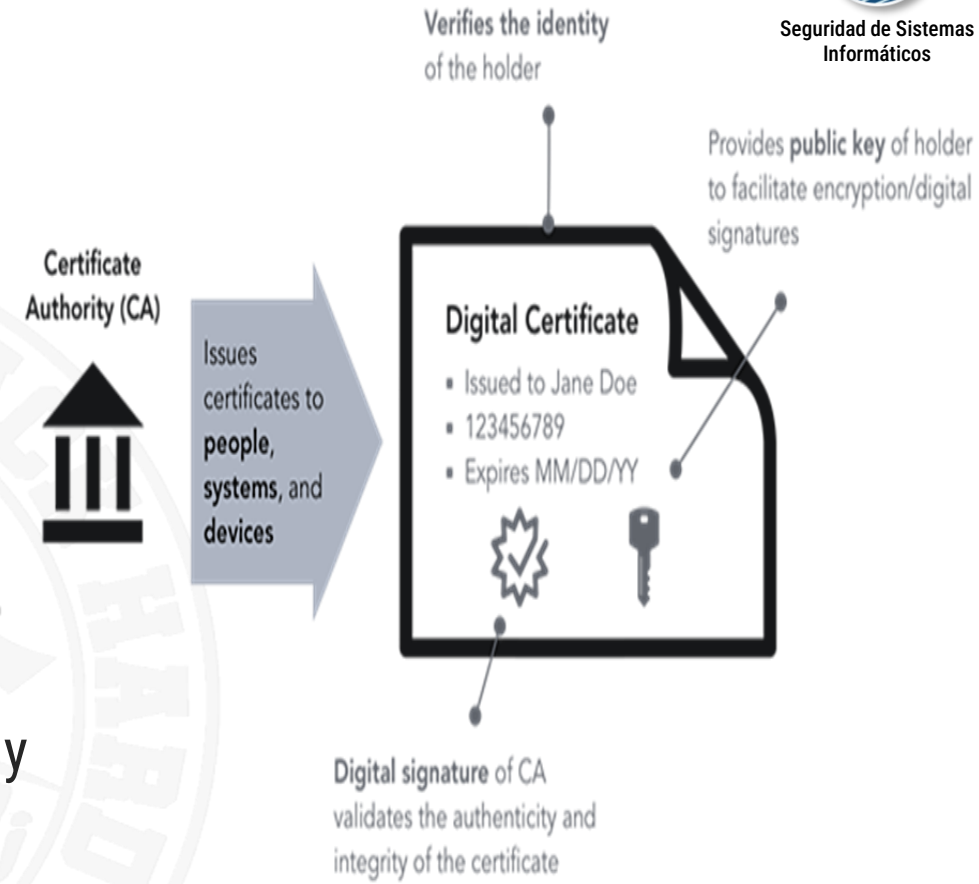
Digital certificates

A digital document to identify them all

*Icons by Bing Chat AI

DIGITAL CERTIFICATES

- **They are used to authenticate public keys that are transmitted between entities**
  - Asymmetric cryptography cannot work if we cannot ensure that a certain public key belongs to a concrete individual
- **A certificate is like an identity card: it is used to identify entities (machines, users…)**
  - They are documents containing **information about its owner and its public key**
  - Part of the public-private key pair in asymmetric cryptography
- **Therefore, a certificate is the association between a physical entity and a transmitted public key**
  - And validated by a trusted entity (E. g.: CA)
- **How is the trust of this association guaranteed?**
  - Ensure non-fake identities
  - Provide non-corrupt keys



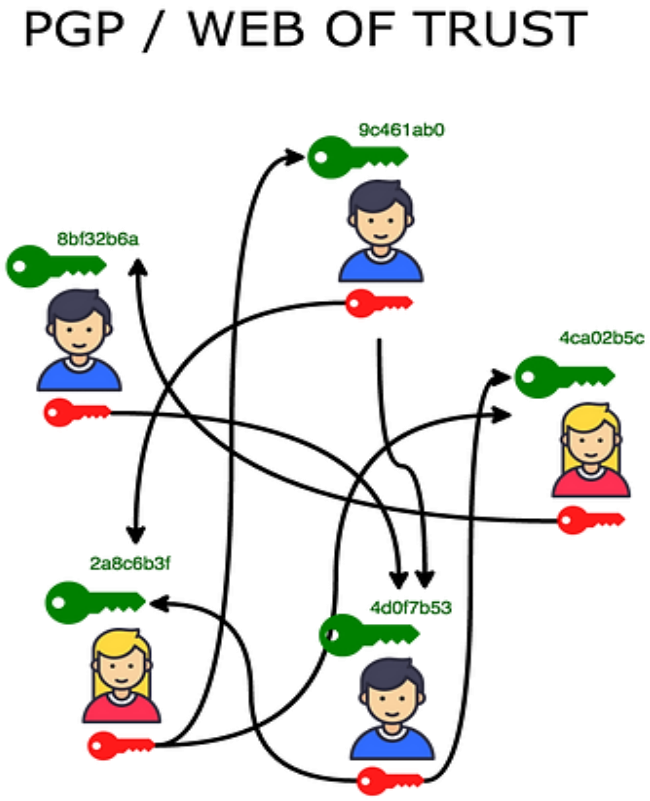
There are certificates for transmitting public keys that only contain this key (e.g., .crt), but there are others that are used to sign inside a machine, and that contain both (e.g., .pfx). Therefore, they should not be passed on to a third party without care, because we would be giving them our private key!.
Source: https://id4d.worldbank.org/guide/digital-certificates-and-pki

User key authentication

- We need an additional mechanism (a 3rd entity) to make sure that the holder of a key is who it claims to be
- There are two, and both follow a trust-based approach
  - An entity U1 **digitally signs** (as previously shown) the certificates (keys) of another entity U2
  - We trust U1, so we also accept that the U2 certificate is valid (trust is "transitive")
- These mechanisms can be
  - **Trusted rings** (e. g. **GnuPG**): Certificates are trusted **by other users**
    - The more users trust a ¡certificate, the more "trustable" it is
      - Like Twitter posts / YouTube videos…but also has its same problems
    - It is a **decentralized**, fault-tolerant trust model
  - **Certificate Authorities (CAs)**: entities that are responsible of establishing certificate validity
    - They work like a **notary**
    - Compromising or disabling a CA invalidates or prevents verification of its associated certificates

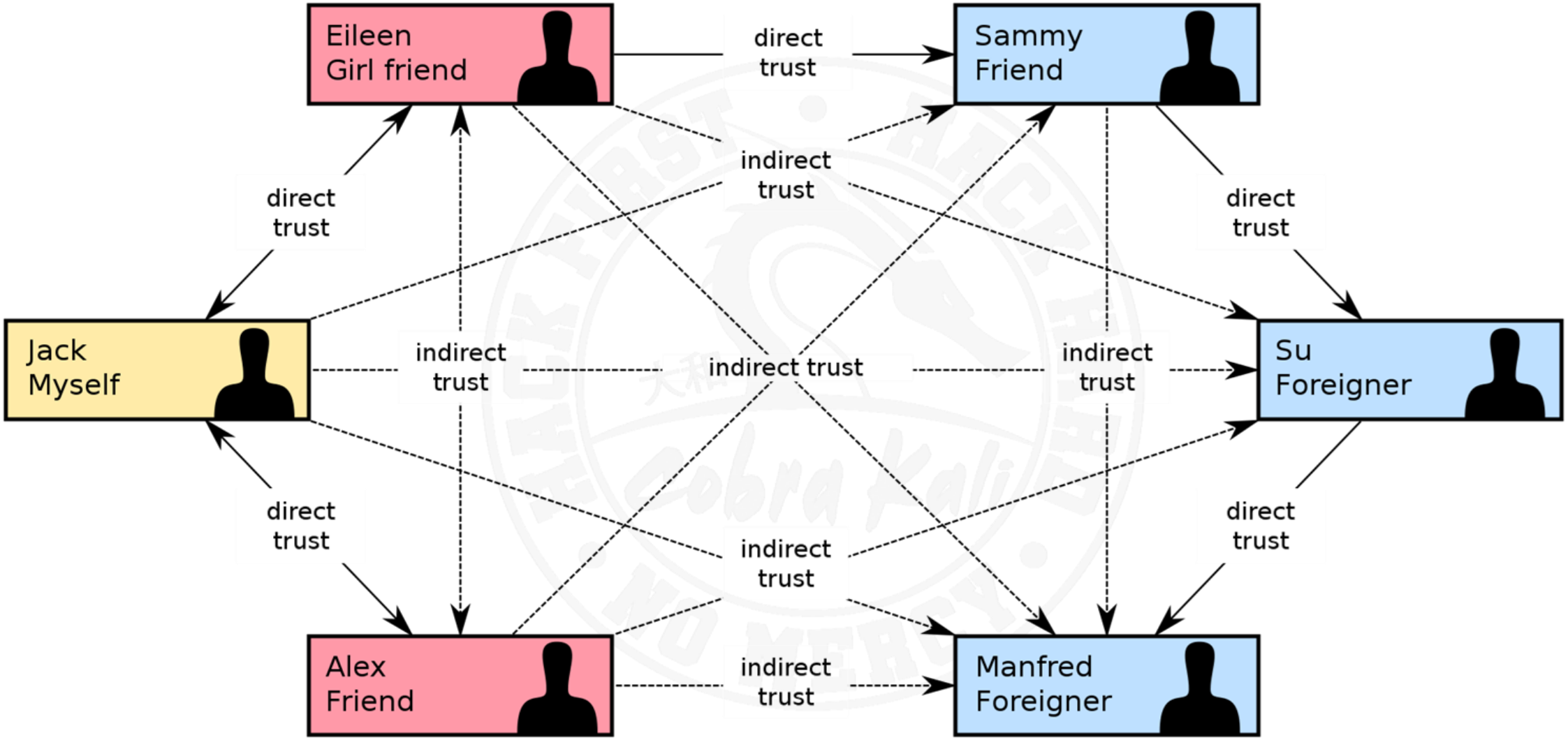
Trusted rings (GnuPG)

- **When we get a public key from a new user, we can associate to it a trust level**
  - **Unknown**. We don't know what trust level to give to a key
  - **None**. The key does not give us any trust level
  - **Marginal**. We're not sure, we know the risks, but we accept the key
  - **All**. The signatures made with that key are as good as ours
    - We trust the key
- **How do we assign a trust level to that key?**
  - The key may be signed by other users
  - The new user will be trusted depending on our trust on those signatures
    - Even without knowing him
  - Trust relationships between a set of users form a **trusted ring**
  - **More info:** https://cran.r-project.org/web/packages/gpg/vignettes/intro.html



Source: https://cran.r-project.org/web/packages/gpg/vignettes/intro.html

Trusted ring (Web of trust)

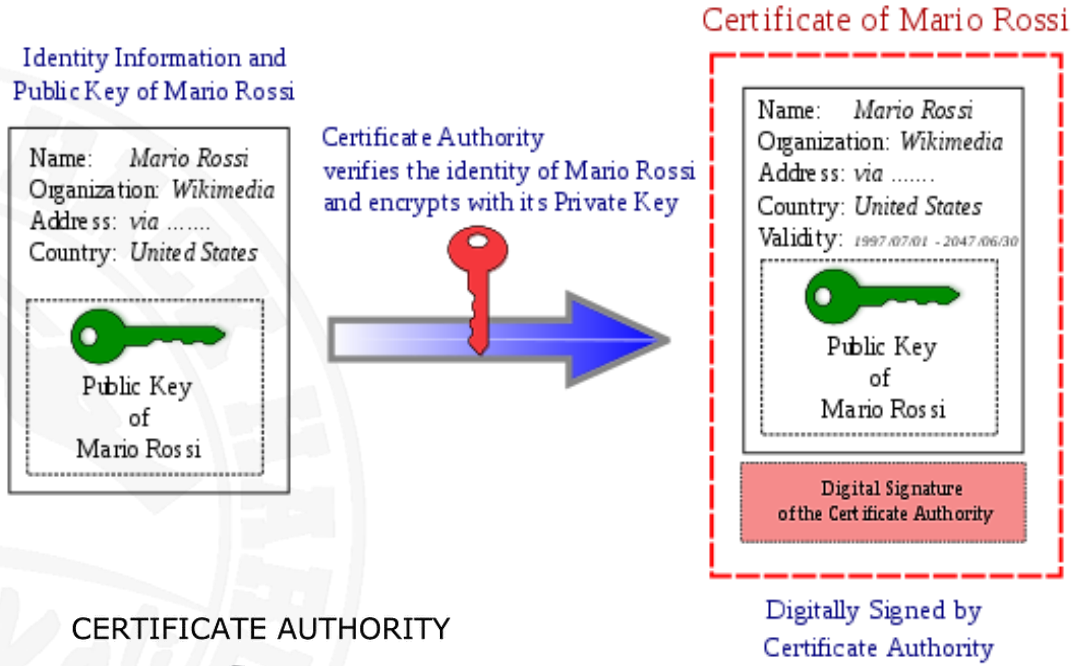


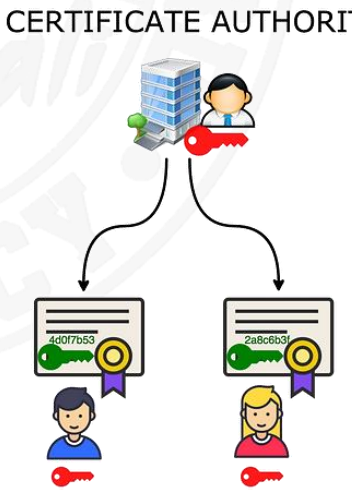
https://en.wikipedia.org/wiki/Web_of_trust#/media/File:Web_of_Trust-en.svg

Certification Authority (CA)

- **They sign** their certificates with its own **private key** to ensure its authenticity
  - It acts like a **notary**, certifying the validity of its certificates
- **If the recipient trusts the CA**, then the certificate information can be trusted
  - And that the sender of the message is who he/she claims to be
- **Additionally, they support applying a hash function to the certificate itself**
  - And to the content of the message
  - This prevents certificate forgery and checks its integrity

Source: https://en.wikipedia.org/wiki/Certificate_authority





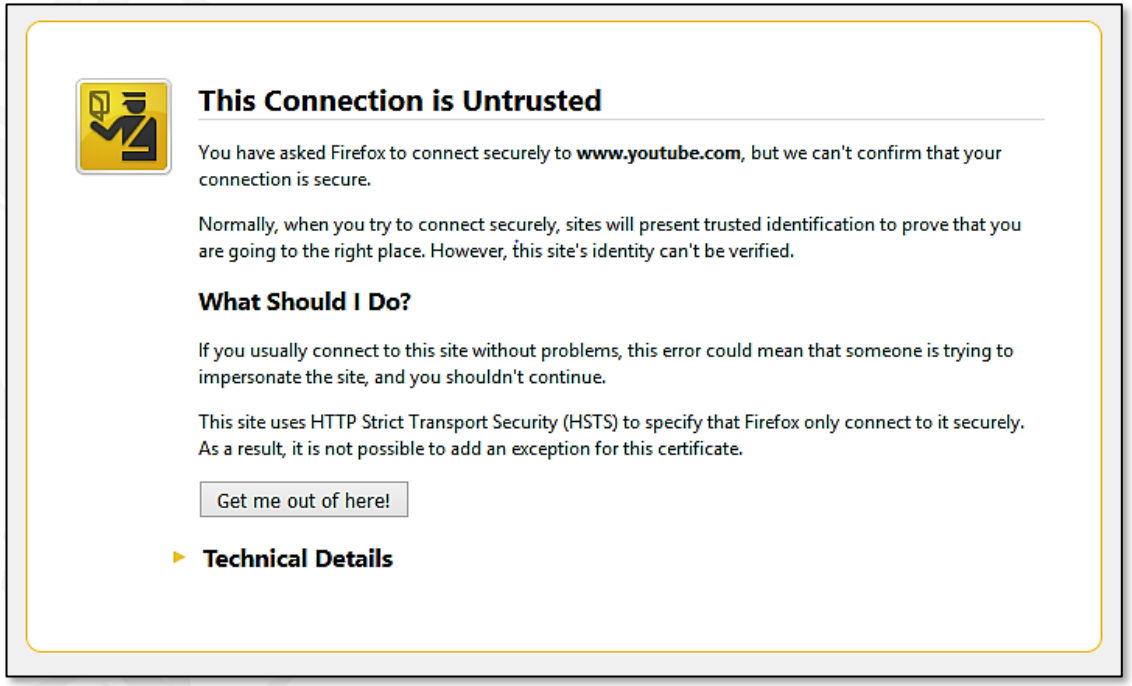
Source: https://cran.r-project.org/web/packages/gpg/vignettes/intro.html

Certification Authority (CA)

- In this trust model, **a certificate will be as reliable as its issuing authority**
- There are several types of CAs
  - **Known CAs** (Verisign, Thawte, ...)
    - These issue certificates at customer request, but typically **at a cost**
    - However, there are very popular free, trusted and known CAs, like Let's Encrypt: https://letsencrypt.org/
  - **Private certifying authorities of an organization**: with the added cost of maintaining them
    - Entities outside the company may not trust it
  - **Public state CAs**: Like the Spain's DNI
    - https://www.dnielectronico.es/PortalDNIe/
- Each certificate must be signed by only one certification authority

Untrusted CAs

- **If we use non-trusted CAs we will see warnings or errors**
  - Browsers include their own list of trusted CAs
  - Using one not in this list (like our own one) gives us this warning
  - Typically, this happens when implementing HTTPs with a self-signed certificate
  - **Common in testing environments, but not suitable for production at all**
- **Information is still signed / ciphered**
  - The problem is that the certificate involved in the process cannot be verified
  - Do you trust the issuer?



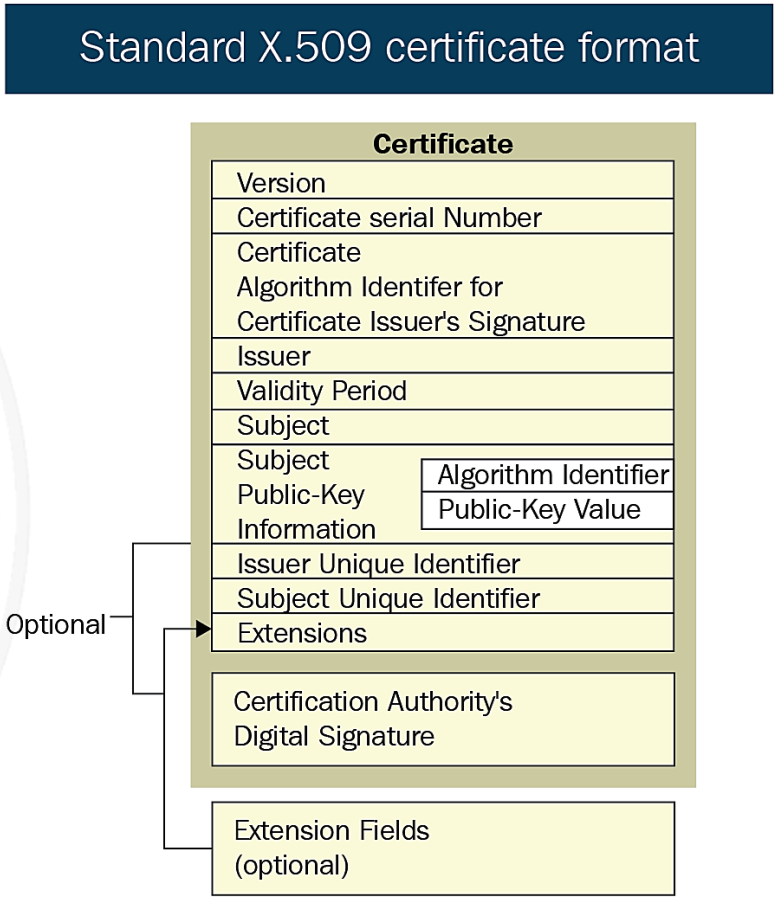
Source: https://support.mozilla.org/en-US/questions/1091062

Digital certificates: Types

- **Personal**: Issued to users for tasks like authentication or digital signatures
  - A single user may have multiple ones, each one signed by the CA that issued it
  - Personal certificates usually may have additional uses
    - **Company membership**: In addition to person identity, certifies that they belong to a certain company
    - **As a company representative**: Certifies membership + company representation powers to the owner
    - **As a legal entity**: identifies a company when carrying out administrative procedures
- **Server / machine**: machines may also need to prove their identity to other entities
  - "I am the server you are looking for"
- **Software**: sent to developers to sign their software
  - Prior to production or distribution
  - "This software is signed by Microsoft"
- **Certificate Authority**: Used by CAs to sign the certificates they send to others
  - "(Luke) I am your CA"

DIGITAL CERTIFICATES: STRUCTURE

- **X.509** is a standard that defines the format of public key certificates
  - Used in many Internet protocols
  - Including TLS/SSL, which is the base of HTTPS
- **All certificates, regardless of type, have the same structure that includes at least**
  - Identity of the certificate owner
  - Public key of the owner
  - Certificate purpose
  - Certificate Issuer Identity (CA)
  - Expiry date
  - Algorithms used
  - Fingerprint (to prove its integrity)
- **Let's see some certificate usages**



Source: https://subscription.packtpub.com/book/business/9781788832687/3/ch03lvl1sec31/pki-certificate-standards-for-iiot

IDENTIFYING SERVERS

Seguridad de Sistemas Informáticos

- Most frequent application
- Ensure that the machine we connect to is really who it claims to be
- The most popular usage is to connect to web servers through the HTTPs protocol
  - **Transmission is also encrypted**: confidential data cannot be obtained by sniffing the network
- Key to ensure
  - Password privacy, secret messaging, electronic transactions...
  - **All our digital life**!

Google signs with a SHA-256 algorithm

Subject Name

Common Name	www.google.com

Issuer Name

Country	US
Organization	Google Trust Services LLC
Common Name	GTS CA 1C3

Validity

Not Before	Tue, 02 Jan 2024 13:09:58 GMT
Not After	Tue, 26 Mar 2024 13:09:57 GMT

Subject Alt Names

DNS Name	www.google.com

Public Key Info

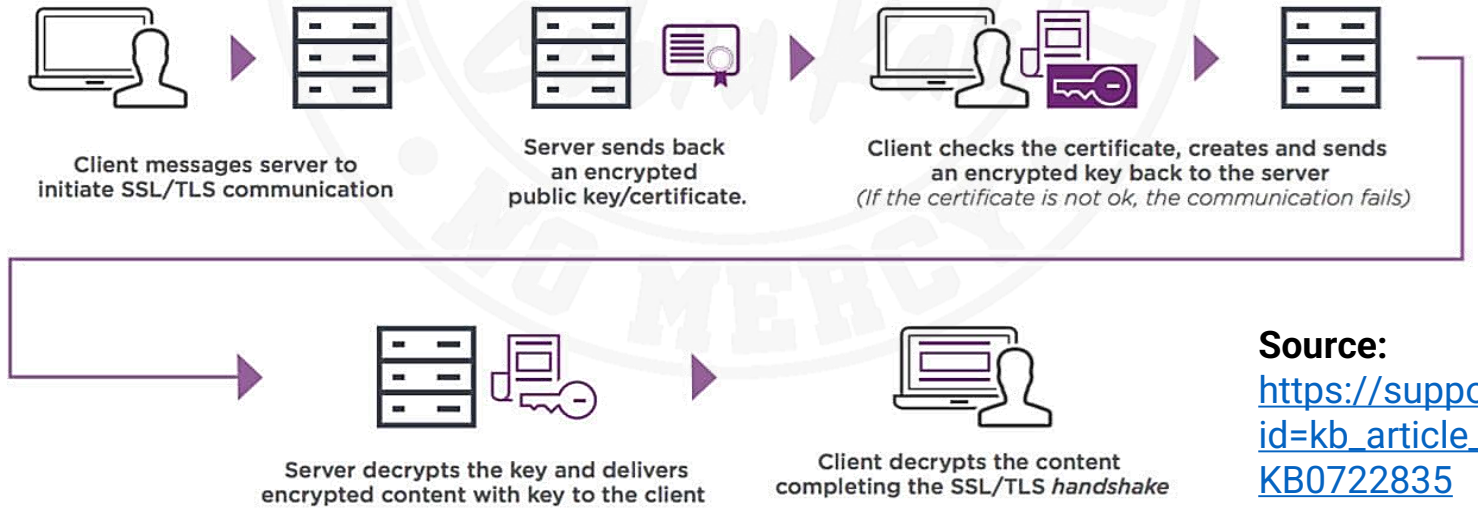
Algorithm	Elliptic Curve
Key Size	256
Public Value	04:62:ED:2D:2D:30:79:FE:94:F1:CC:C8:AE:5F:4F:86:D8:2E:79:AB:16:23:CF:4B:E7:B9:...

Miscellaneous

Serial Number	24:C8:58:2B:4E:AE:25:46:0A:7F:D2:CE:BB:16:80:9A
Signature Algorithm	SHA-256 with RSA Encryption
Version	3

REMOTE TRANSACTIONS: HTTPS

- **Secure HTTP** connections are a transparent certificate negotiation process stablished between browsers and web servers
- Ensures secure data transfer if a secure ciphering algorithm is negotiated
  - Not possible with **old clients** (E. g.: Windows XP)
  - Or with **misconfigured web servers** (cryptographic downgrade attacks)
- Configuring any webserver with adequate HTTPs protocol features is easy following this guide: https://ssl-config.mozilla.org/

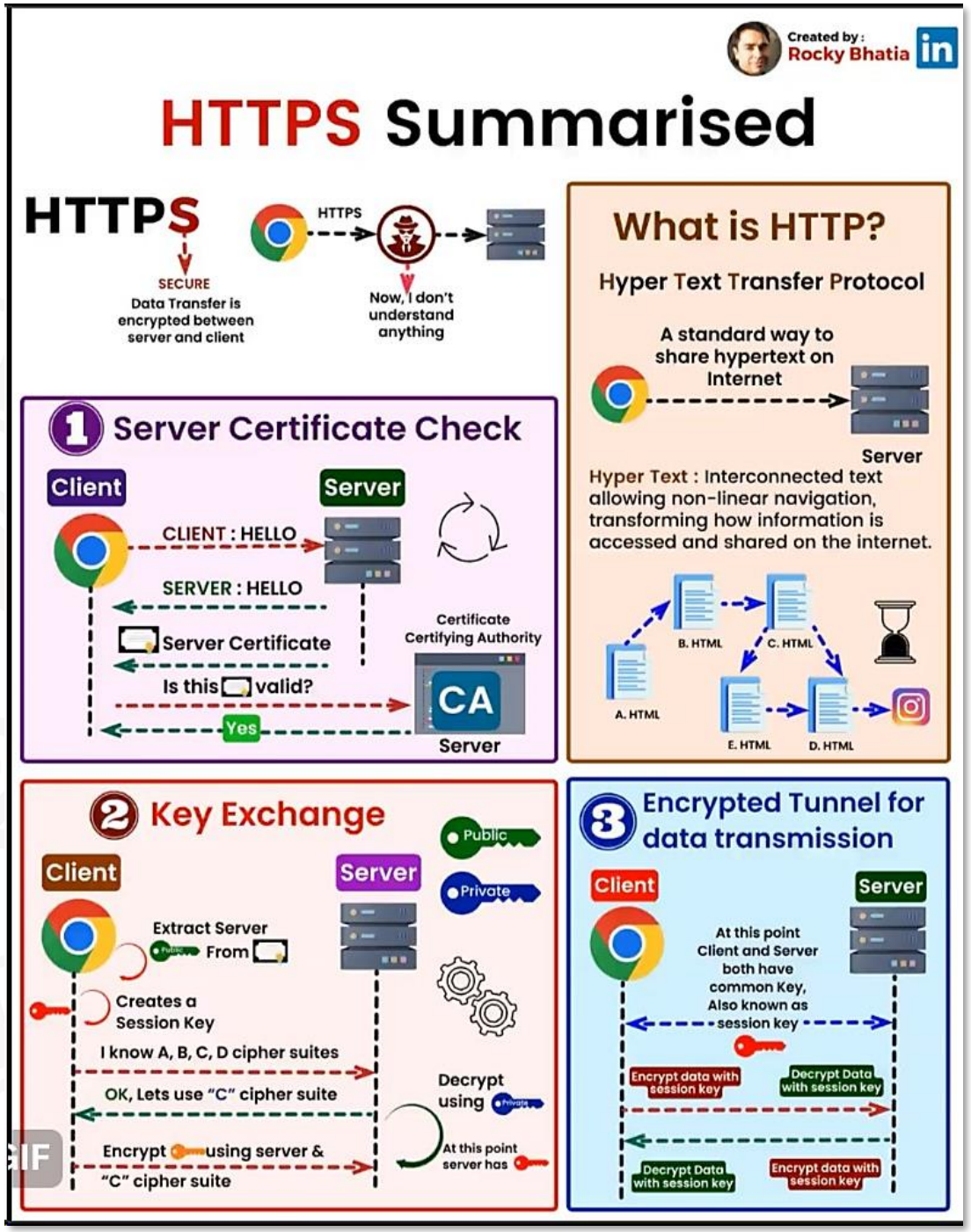


Source:
https://support.servicenow.com/kb?id=kb_article_view&sysparm_article=KB0722835

Remote transactions: HTTPs

Seguridad de Sistemas Informáticos

- **As you see, a lot of concepts we reviewed are involved in an HTTPs connection**
  - **Certificates** to identify servers and transmit its public key to the client
  - **Asymmetric encryption** to cipher a per-connection symmetric encryption key
    - That also ensures its secure transmission from client to server
  - **Symmetric encryption** to all sent data
    - Faster than using asymmetric encryption to do it
    - Do you realize that now **both share a symmetric key**? (session key)
      - That's when AEAD comes to play
      - Ciphered and authenticated traffic!

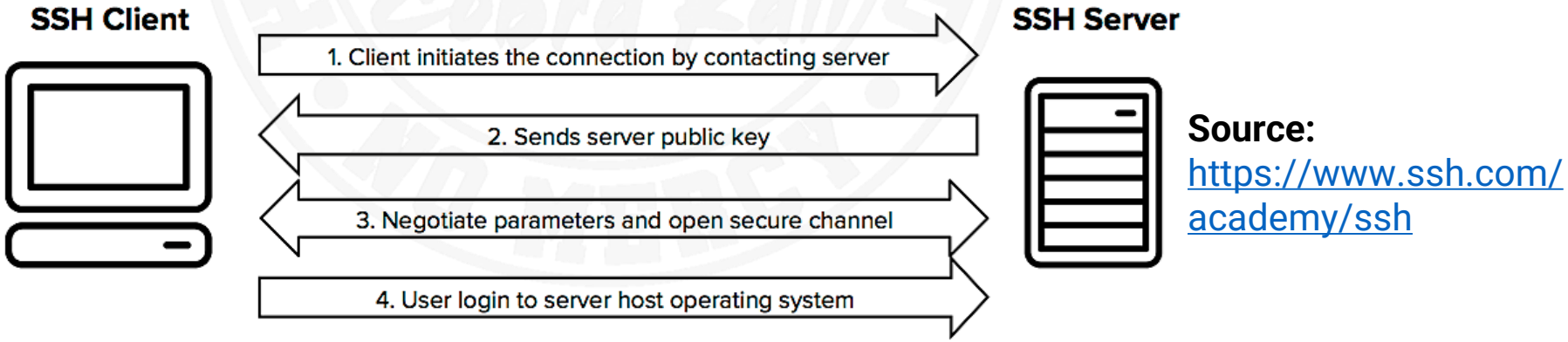


REMOTE TRANSACTIONS: SSH

- **You can use SSH to encrypt connections to a remote machine**
  - When connection starts, the server sends its public key to the client
  - If the client accepts this key, it stores it for future connections

```
The authenticity of host '10.0.0.1 (10.0.0.1)' can't be established
but keys of different type are already known for this host.
RSA key fingerprint is f8:88:86:4f:d7:9f:8c:c2:a2:e9:0b:01:8f:5b:8e:fd.
Are you sure you want to continue connecting (yes/no)? _
```

  - The client generates a **random session key**, encrypts it with the server public key, and sends it back
  - This key will be used for encrypted transmission of the data (**symmetric encryption**)
    - **Remember**: it is **faster** than public key encryption
- **Configuring a properly secured SSH server is easy following these guidelines**
  - https://infosec.mozilla.org/guidelines/openssh

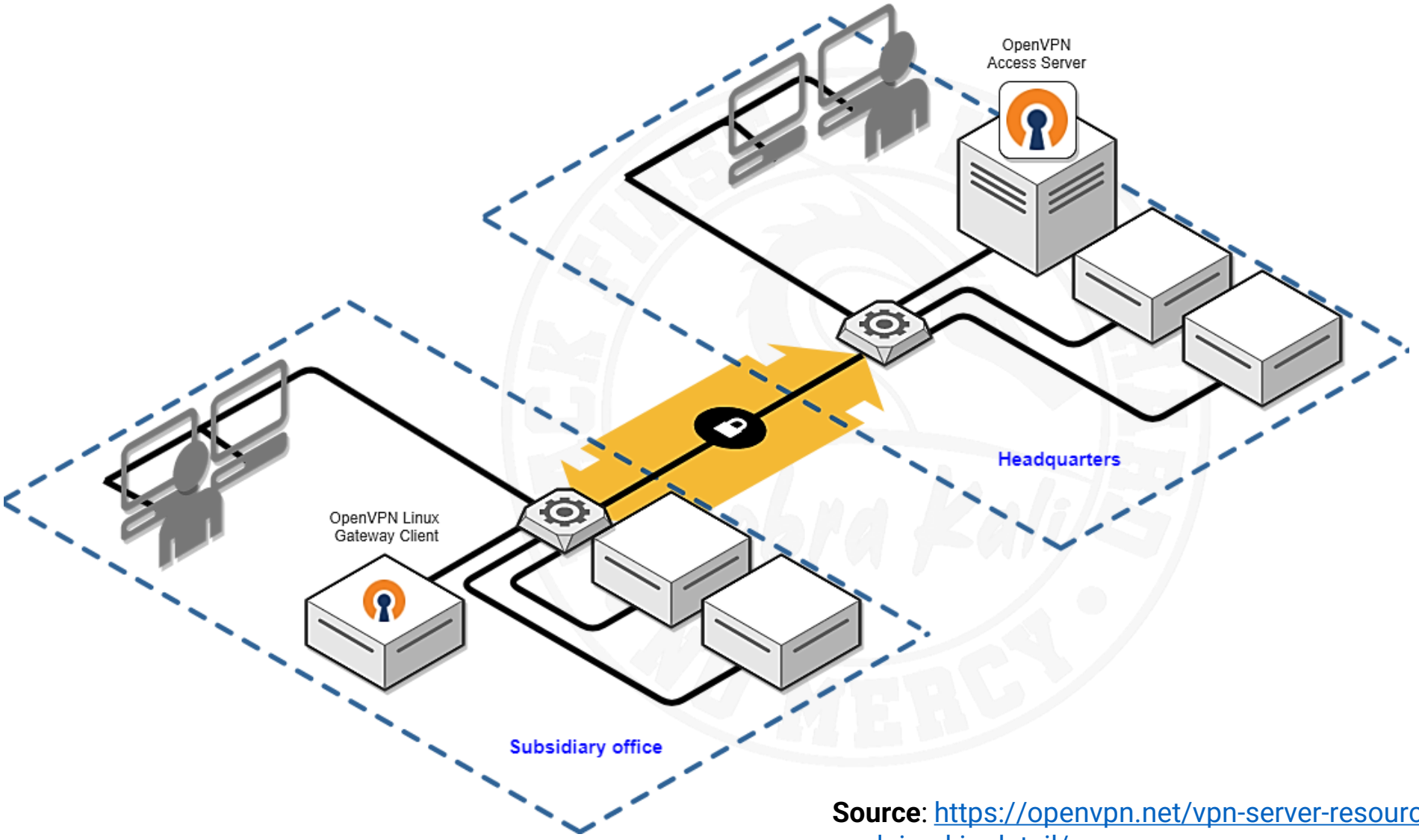


Source: https://www.ssh.com/academy/ssh

Virtual Private Networks (VPNs)

- **Extends a private network across a public network**
  - Created by establishing a virtual point-to-point connection using dedicated circuits or with tunneling protocols over existing networks (the Eurotunel can be a good metaphor ☺)
  - From a user perspective, the resources within the private network can be accessed remotely
  - Enables users to exchange data across shared or public networks **as if their devices were directly connected to the private network**
- **VPN technology was developed to allow remote users and branch offices to access corporate applications and resources**
  - To ensure security, a private network is established using an encrypted layered tunneling protocol
  - VPN users use authentication methods, including passwords or **certificates**, to access to it
- **Popular implementations**
  - WireGuard is a highly recommended VPN software: https://www.wireguard.com/
  - ZeroTier, a P2P VPN (we will see what this is ☺): https://www.zerotier.com/
  - ExpressVPN: Is the TechRadar 2020 and CNET Editor's Choice: www.expressvpn.com

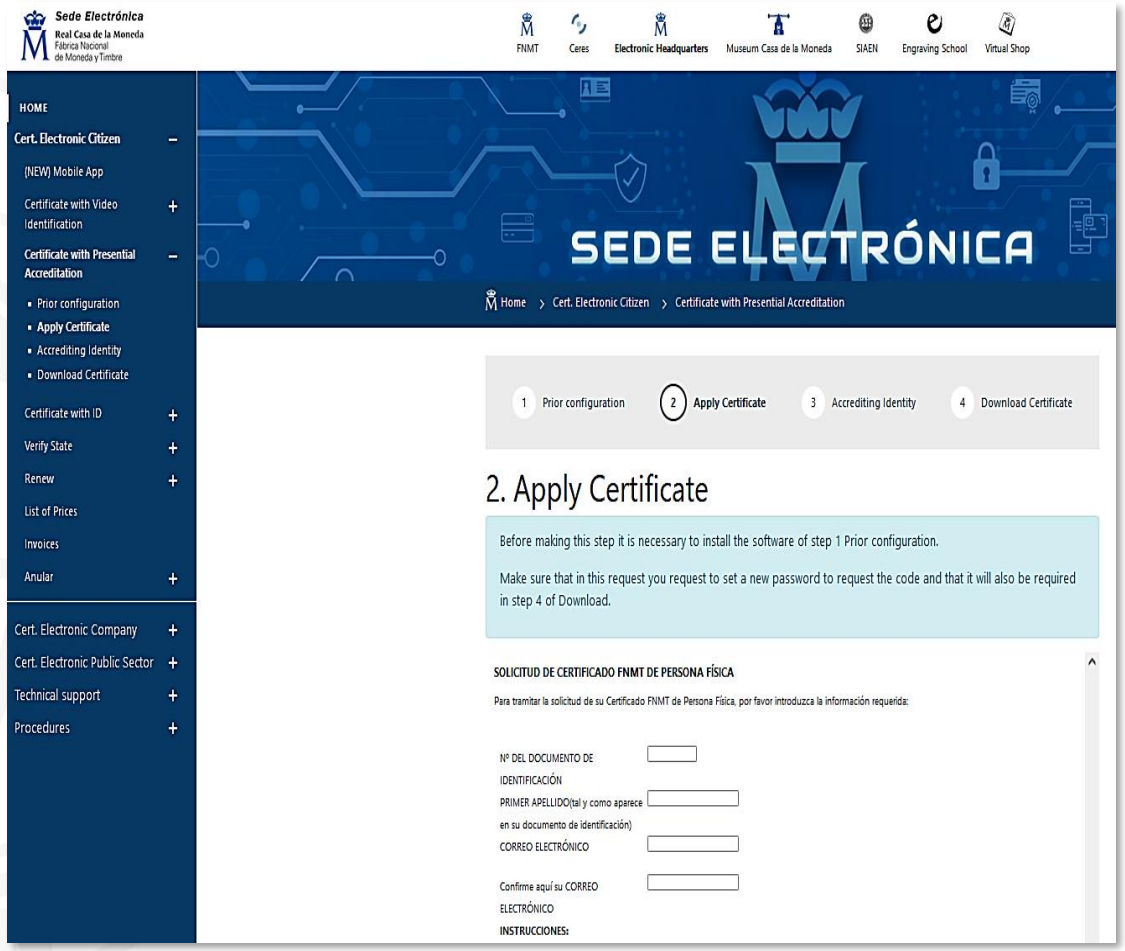
Traditional Virtual Private Networks (VPNs)



Source: https://openvpn.net/vpn-server-resources/site-to-site-routing-explained-in-detail/

REMOTE TRANSACTIONS: CUSTOMERS / CLIENTS / CITIZENS

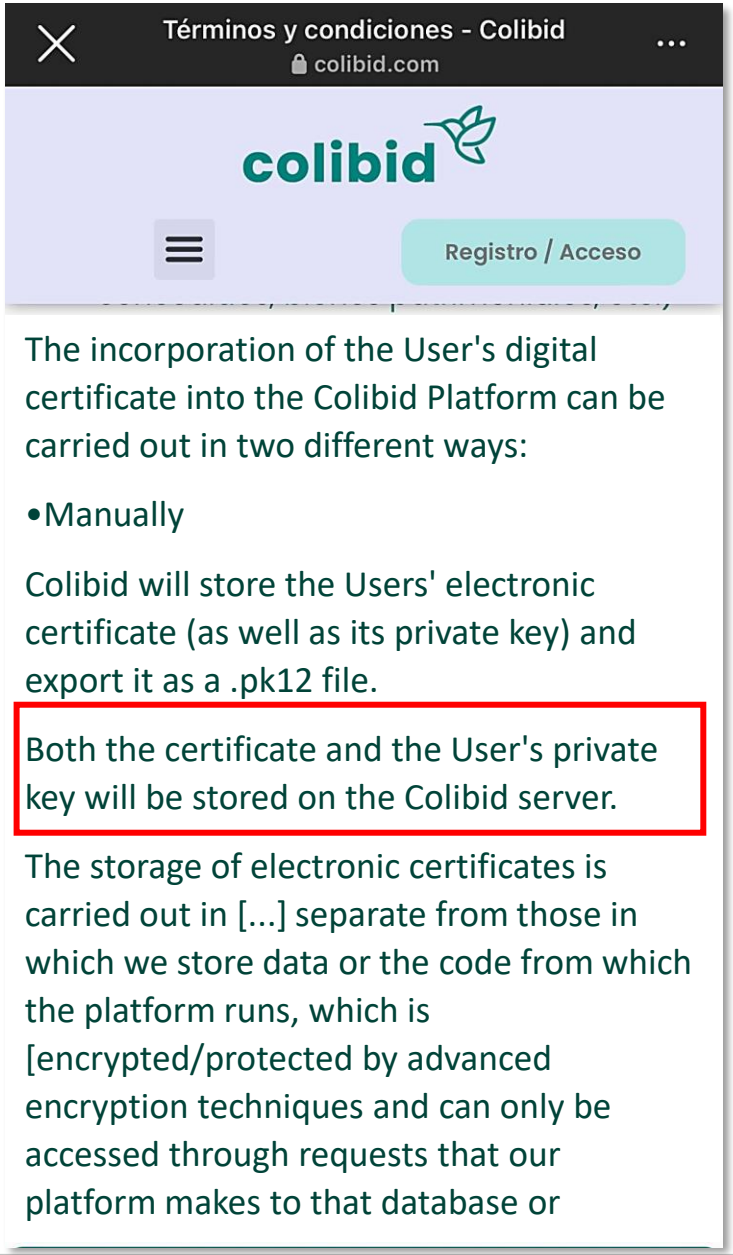
- **Certain administrative procedures require users to identify themselves**
  - The company / institution **MUST** ensure that the person doing the operation is who he claims to be!
- **This is especially important**
  - In official procedures: Tax Agency, Town Halls...
  - And for commercial transactions
- **Certificates are replacing passwords and coordinate cards in these kind of operations**
- **Spanish official institutions recognize certificates issued by the Ceres**
  - From the National Mint and Stamp Factory
  - http://www.cert.fnmt.es/



Source: https://www.sede.fnmt.gob.es/certificados/persona-fisica/obtener-certificado-software/solicitar-certificado

Remote transactions: Customers / clients / citizens

- **You must be very careful with the use of certificates that represent you legally**
  - Where do we store them and who do we give them to
- **For example, this application is asking the user to give it a certificate of their identity with both keys**
  - Now, in practice, the application will be able to carry out any electronic procedure on behalf of the user without restrictions
  - The application goal is to carry out these procedures...
  - ... But it is asking its users to blindly trust it when it comes to keeping and protecting the certificates
- **If its certificate store is compromised, the attacker could impersonate all its users in e-government procedures directly!**

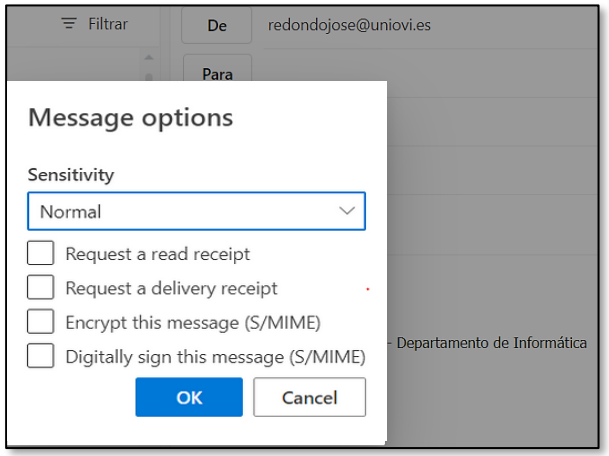


These polemic terms and conditions have since been modified to not to require such sensitive information of the user

EMAIL / FILE AUTHENTICATION

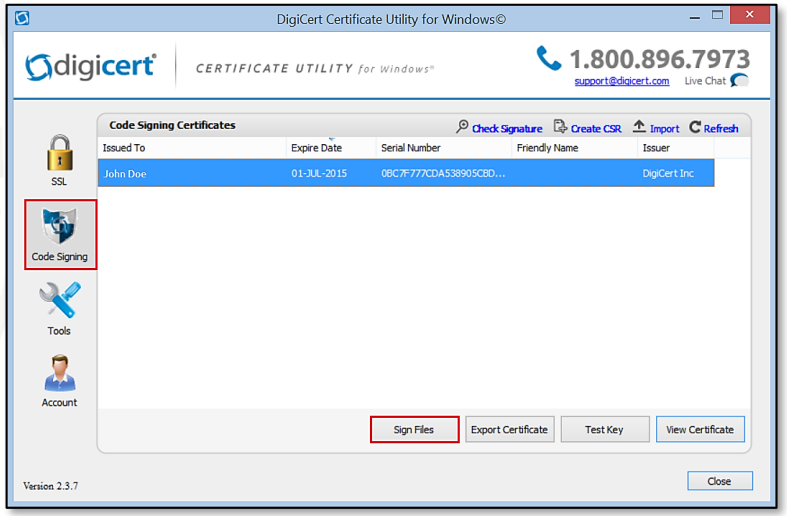
Emails

- Certain mail clients allows sending and receiving signed and / or encrypted emails
  - There are systems based on RSA or GnuPG certificates
  - https://support.microsoft.com/en-us/office/encrypt-email-messages-373339cb-bf1a-4509-b296-802a39d801dc
  - https://support.microsoft.com/en-us/office/secure-messages-by-using-a-digital-signature-549ca2f1-a68f-4366-85fa-b3f4b5856fc6
- Supported by certain webmail services (e. g. ProtonMail)
  - Browsers support digital certificate management, so the server may ask the browser for the user's identity



Files

- There are applications that allow to sign and / or encrypt files with RSA/GnuPG or similar
- Allow using electronic documents with the same legal validity as signed physical documents
- There are many systems, but we will have to use those accepted by our institutions / companies (eCofirma, Adobe...)

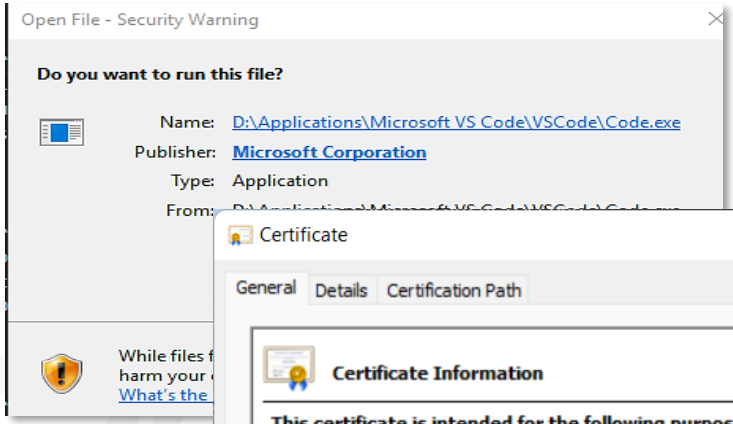


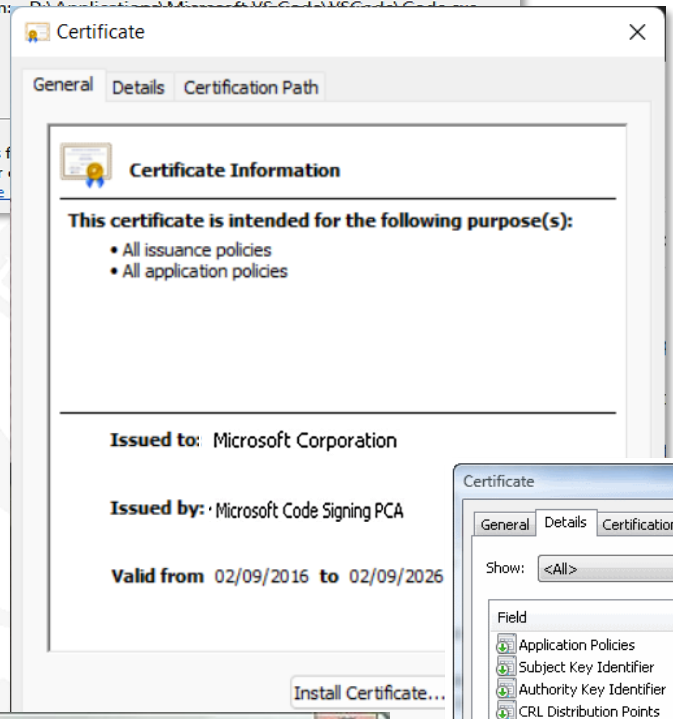
Source: https://www.digicert.com/kb/code-signing/digicert-certificate-utility-to-sign-code.htm

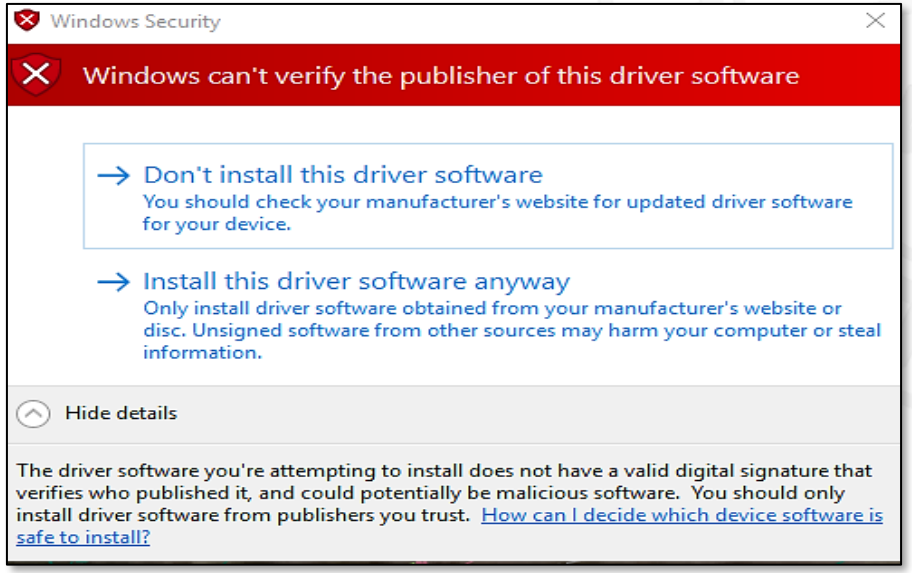
➢ Exercise **L4B3_CERTSIGNPDF**, **L4B4_CHECKSIGN**

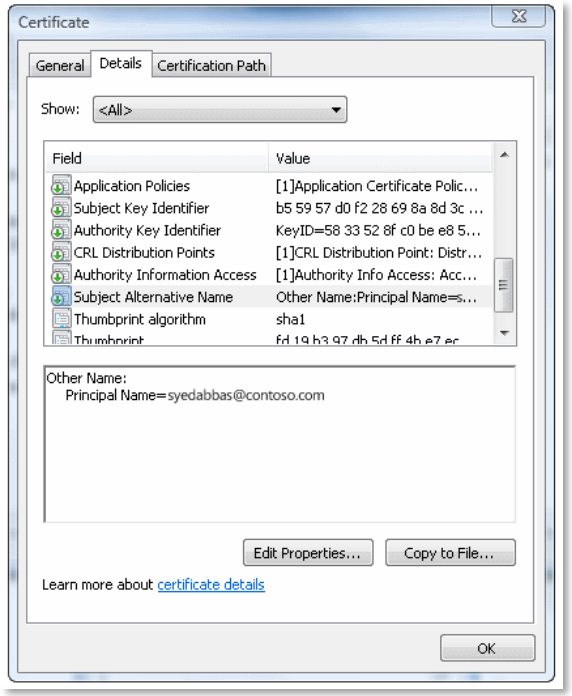
CODE AUTHENTICATION

- Programs can be signed to **ensure its authorship and content integrity**
  - https://en.wikipedia.org/wiki/Code_signing
- A program can have 3 possible states
  1. **Unsigned**. Author is unknown
  2. **Signed**, but **without** a trusted certificate
  3. **Signed with** a trusted certificate









Think you've understood it all? Evaluate yourself!

?

- **You can do the following**
  - *You understand that, in reality, a digital certificate is nothing more than a document with a known structure that says that a certain key(s) belong to a known person or entity*
  - *You understand that there are certificates designed to be sent to anyone (with your public key) and others designed to stay on a specific machine (with both keys),*
    - *And that in case of transmitting the latter, you must do so very carefully and to trusted places (e.g., other machines you own)*
  - *You understand the important role that rings of trust and certificate authorities play when working with certificates*
    - *And the differences between the two approaches*
  - *You can remember the main applications of digital certificates*
  - *You understand how some of them combine symmetric and asymmetric encryption to take advantage of their main advantages*

< Go to Index

APPENDIX

Other interesting concepts

*Icons by Bing Chat AI

Source: Bing Chat AI

PLAUSIBLY DENIABLE ENCRYPTION TECHNIQUES

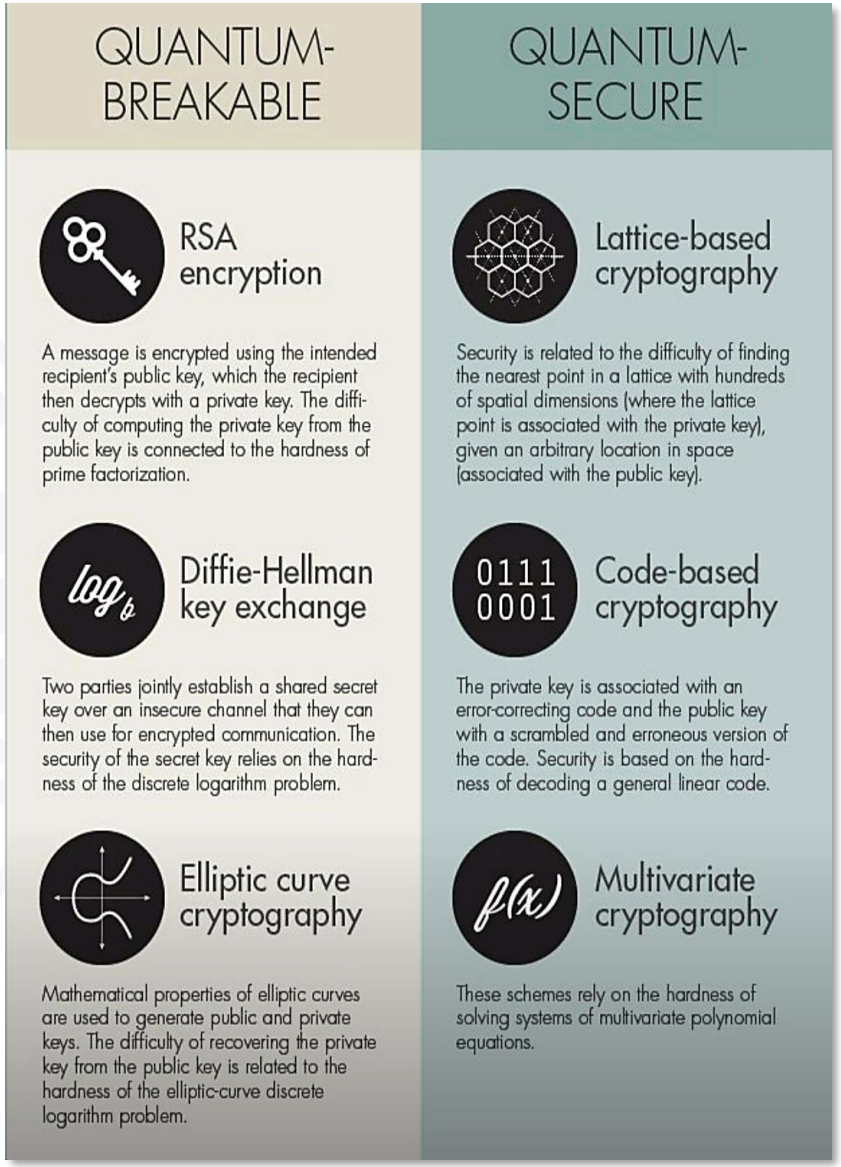
- **The existence of an encrypted file or message is deniable**
  - An adversary cannot prove that the corresponding plaintext data exists
- **Deniable encryption makes it impossible to prove the existence of the plaintext message without the proper decryption key**
  - This may be done by **allowing an encrypted message to be decrypted to different plaintexts**, depending on the key used
    - **False data** (decoy): give the adversary false data after apparently deciphering the data correctly
    - **Random data** (chaff): give the adversary the sensation that there is no real ciphered data
  - This allows the sender to do a plausible deniability if forced to give an encryption key
- **This concept was used by Julian Assange and Ralf Weinmann**
- **There are some tools that facilitate deniable encryption**
  - OpenPuff Steganography and Watermarking
  - FreeOTFE (On The Fly Encryption)
  - TrueCrypt, LibreCrypt, and BestCrypt

QUANTUM CRYPTOGRAPHY

- **Uses principles of quantum mechanics to exchange random keys in insecure channels**
- **It is not (currently) breakable, key exchange is secure**
  - If someone listens during secret key creation, **the process is altered**
  - Warning communicating parties before information is transmitted
  - This is a consequence of Heisenberg's principle of uncertainty
    - Measuring in a quantum system disrupts it (one of it most important properties)
- **Quantum Key Distribution protocols have been developed**
  - Their implementation have a sizable cost…
  - Since 2007, the transmission of the vote count in the Geneva Federal elections is protected with it
  - Some prestigious Swiss banks are also using it
  - Lasers are used to emit through optical fibers information on the constituent element of light (the photon)

OTHER CRYPTOGRAPHY CONCEPTS / APPLICATIONS

- **Cryptography "trending topics"**
  - **Homomorphic encryption**: allows computation over ciphered data
  - **Lightweight encryption**: for devices with low computational power, like IoT
  - **Whitebox encryption**: protection of keys while they reside in RAM
  - **Post-quantum encryption**: algorithms that can be executed in traditional computers…
    - But robust against quantum ones!
  - **Searchable encryption**, **Differential cryptography**, **Format-Preserving Encryption** (FPE)…
- **You want to use cryptography effectively?**
  - Cryptographic best practices: https://gist.github.com/atoponce/07d8d4c833873be2f68c34f9afc5a78a



Source: https://es.linkedin.com/in/ismaelr

To know more…

- If you are passionate about this field, try these books
  - Practical Cryptography for Developers
    - https://cryptobook.nakov.com/
  - Handbook of Applied Cryptography
    - https://www.amazon.es/Handbook-Cryptography-Discrete-Mathematics-Applications/dp/0849385237
  - Real-World Cryptography
    - https://www.manning.com/books/real-world-cryptography
  - Foundations of Cryptography
    - Volume 1, Basic Tools: https://www.amazon.es/Foundations-Cryptography-Basic-Tools-Vol/dp/0521791723
    - Volume 2, Basic Applications: https://www.amazon.es/Foundations-Cryptography-Basic-Applications-Paperback/dp/052111991X
  - Serious Cryptography: A Practical Introduction to Modern Encryption
    - https://www.amazon.es/Serious-Cryptography-Practical-Introduction-Encryption/dp/1593278268
  - Resources from the prestigious cryptography researcher Alfonso Muñoz
    - Books "Seguridad en el protocolo SSL-TLS" and "Esteganografía lingüística y canales encubiertos" (free): https://github.com/mindcrypt/libros
    - But, if you like cracking ☺…: https://github.com/mindcrypt/cracking
- You want to specialize? Check the **CriptoCert Certified Crypto Analyst** certification
  - But you'll have to study a lot!: https://www.criptocert.com/CriptoCert_Analyst.html

TOPIC 2
CRYPTOGRAPHY APPLICATIONS