TOPIC 2. CRYPTOGRAPHY APPLICATIONS

Protecting your data from leaks & unauthorized access

Source: zona-militar.com

JOSÉ MANUEL REDONDO LÓPEZ *"S-81 ISAAC PERAL"* PROJECT V3.0

Departamento de Informática
Universidad de Oviedo

Universidad de Oviedo

Logo: @creative_vanesa (*Instagram*)

Our goals

José Manuel
Redondo López

- **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**
  - The following are
  - **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…
    - Only allows the contents to be transferred with an acceptable character set
    - They are easily reversable

INDEX

< Go to Index

Source: zona-militar.com

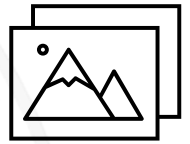
Introduction

What are we talking about?

WHY CRYPTOGRAPHY?

José Manuel
Redondo López

- **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…)
  - **Steganography**: hide the **Existence** of the information
    - Information is **embedded** into another different piece of information (e. g. image)
      - Or turned inaccessible 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



Z$5%tyc…zz

Source: *PowerPoint* icons

CRYPTOLOGY

José Manuel
Redondo López

- **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
  - 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

Cryptography objectives

José Manuel
Redondo López

- **Properly secured communications are basic to achieve real security**
- **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 intended
    - 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 claimed 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
- **Authenticated encryption algorithms provide all three objectives**
  - https://en.wikipedia.org/wiki/Authenticated_encryption

< Go to Index

Source: zona-militar.com

Symmetric >

Asymmetric >

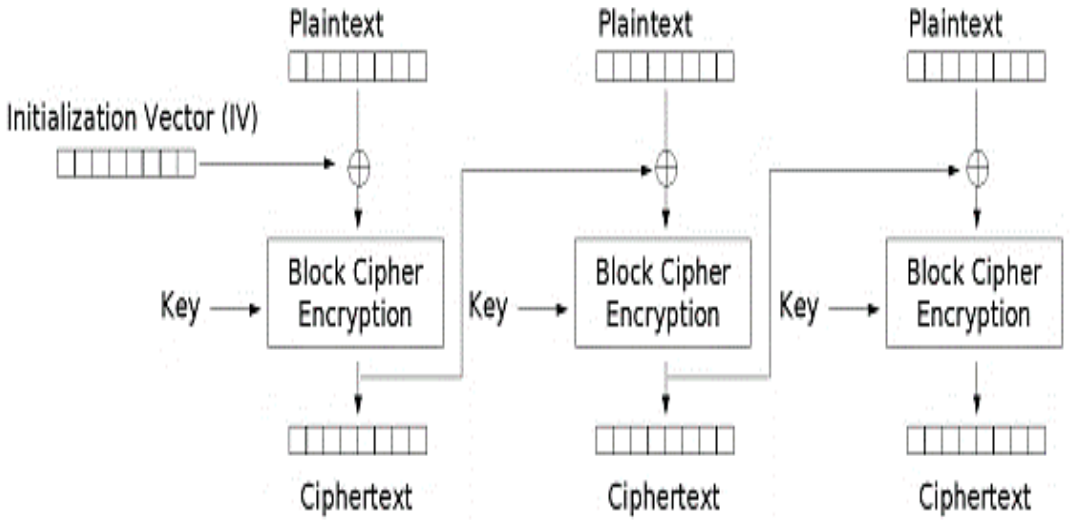
CONFIDENTIALITY

Ciphering algorithms

Classification of ciphering algorithms: how they process the information

José Manuel Redondo López

- **(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 ...
    - 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...
      - 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

José Manuel Redondo López

- **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, … (you can test them in https://cryptii.com/)
- **Cryptographic key ciphers (the ones studied in the course)**
  - Use **keys** (information): 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**
    - **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

José Manuel
Redondo López

- **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 placing 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 C. Shannon's information theory**
  - https://en.wikipedia.org/wiki/Information_theory

Cryptographic key

José Manuel
Redondo López

- **Information pattern that cryptographic algorithms use to cipher and decipher the information**
- **We obtain 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))
```

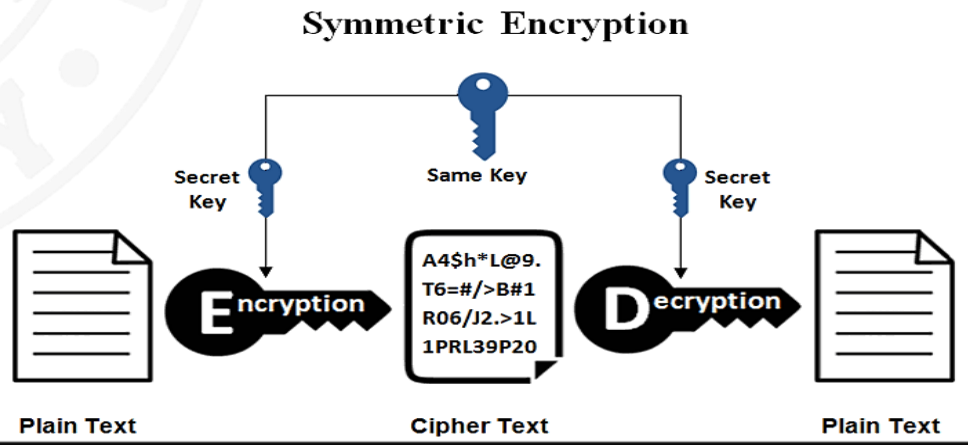
Symmetric-key algorithms

Ciphering with only one shared key

Symmetric-key algorithms (aka private key algorithms)

José Manuel Redondo López

- **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 (non-linear functions, XOR functions…remember the CN and Algebra courses) to make deciphering more difficult
    - 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 K**ey 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!

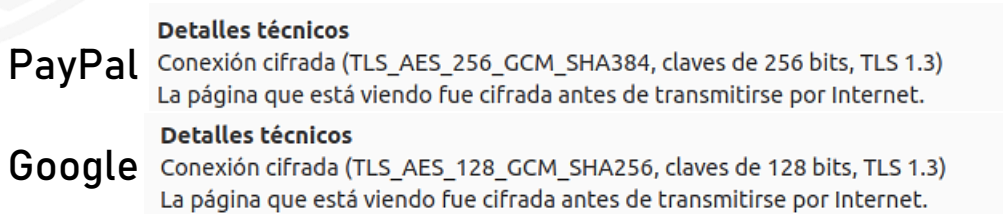


Source: https://cryptobook.nakov.com

SECURE SYMMETRIC-KEY ALGORITHMS

José Manuel
Redondo López

- **An algorithm is considered computationally secure if an attacker cannot break its encryption faster than a brute-force attack**
  - Spending 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**



KEY SIZE MISCONCEPTIONS

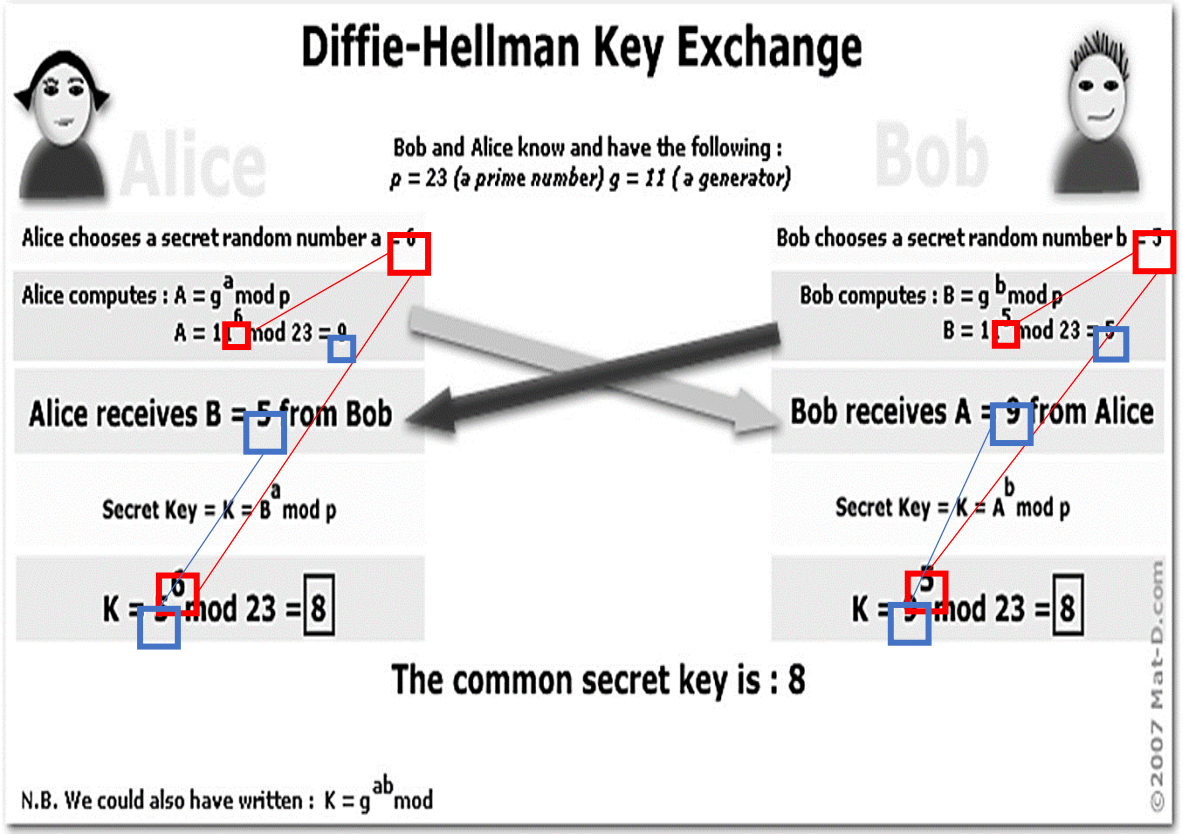
José Manuel
Redondo López

- **Are algorithms with larger key sizes stronger than those using shorter ones?**
  - **NO, the algorithm itself is more important**
  - E. g.: Blowfish supports up to 448-bit keys while AES supports up to 256-bit keys
  - However, the first has been found vulnerable to certain attacks
- **Should I choose the largest available key size of a particular algorithm?**
  - **Not necessarily**, it is not always the optimal decision (remember the information process costs!)
  - The only threat to AES-128 and AES-256 is quantum computers
    - A practical quantum computer may use the Grover's algorithm to break 128-bit (not 256-bit) AES
  - Private-key algorithms requires securely exchanging cryptographic keys over a public channel
    - **Remember**: you must share the key with the other part for the algorithm to work!
    - This is usually done using asymmetric cryptography (Diffe-Hellman key exchange algorithm, see next)
    - https://www.criptocert.com/PDF/CriptoCert_Generacion_claves_RSA_y_numeros_primos_v1.1_RaulSiles.pdf
    - Asymmetric cryptography algorithms (seen later) are usually broken faster
    - Hence, large key sizes do not matter if we have a weaker point

Diffie-Hellman key exchange algorithm

José Manuel
Redondo López

- **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

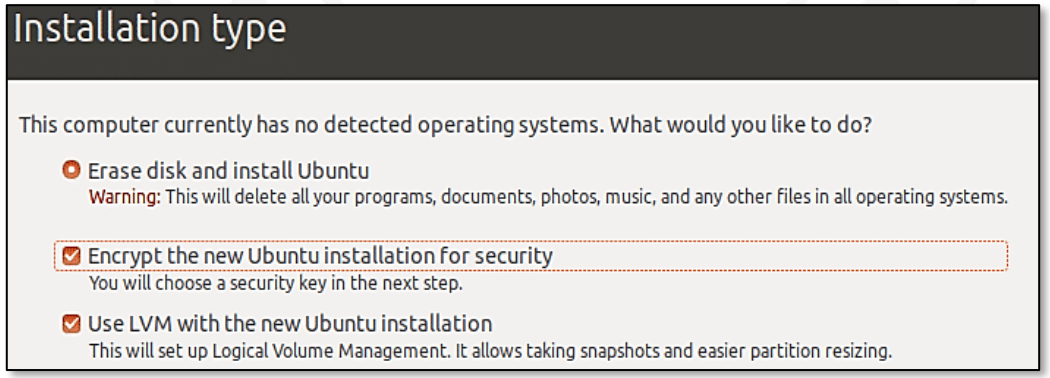
How big is "big enough"?: Real implementations uses 2048-bit numbers (in 2016, the NSA 3072+ bits)

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

Symmetric-key algorithms practical applications: Disk encryption

José Manuel
Redondo López

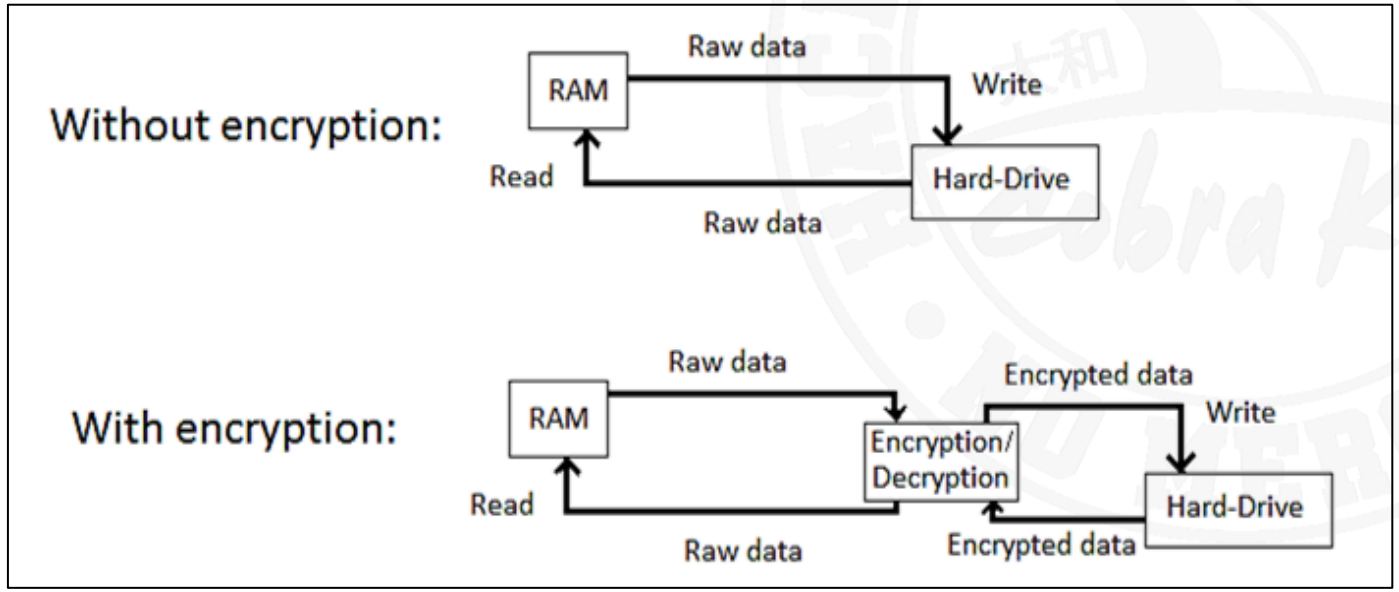
- **Technology to cipher stored information, preventing unauthorized access**
  - Preventing data leaks due to physical hard-drive theft is the most common example
  - Data of an encrypted volume cannot be read (decrypted) without the correct password / key file(s)
- **Uses symmetric-key encryption software / 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 to handle unencrypted data, nor must perform any special operation



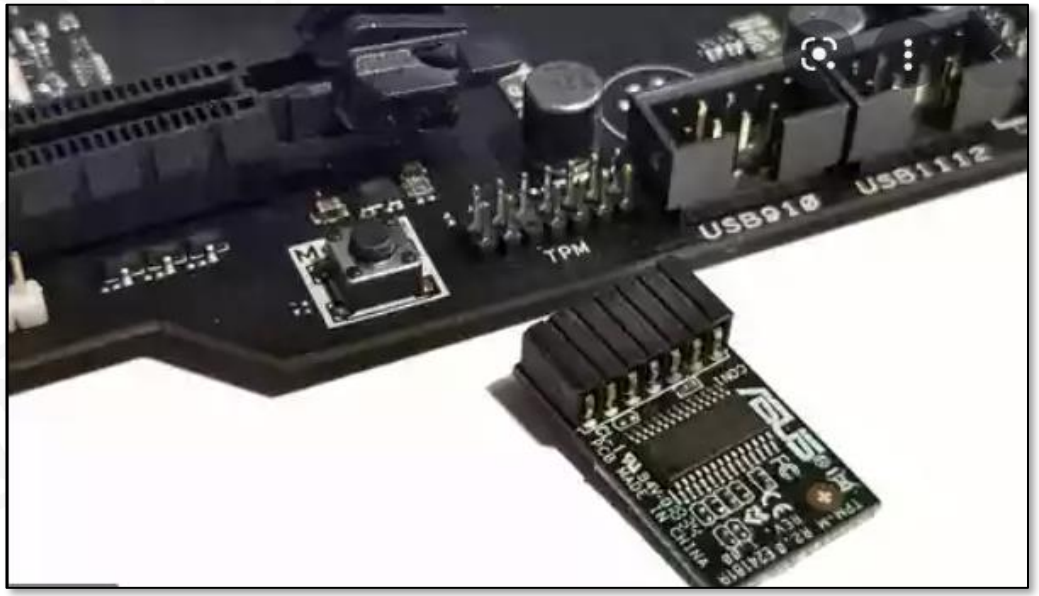
SYMMETRIC-KEY ALGORITHMS PRACTICAL APPLICATIONS

José Manuel
Redondo López

- **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 that the system accessing the device is authorized
  - "Transplanting" an encrypted drive does not work!
  - It is also a requisite to install Windows 11 nowadays



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

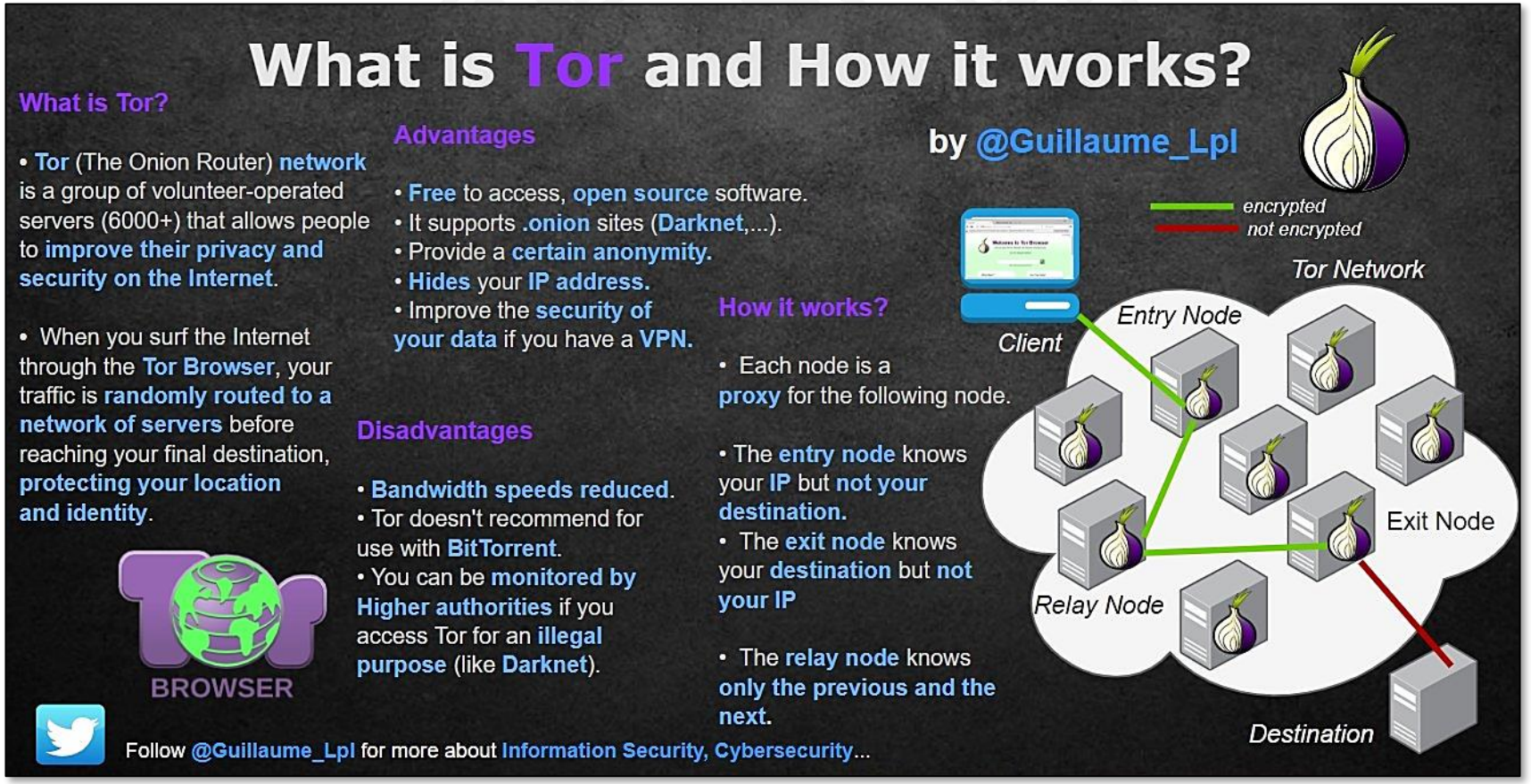


Source: https://hardzone.es/reportajes/que-es/trusted-platform-module-tpm/

SYMMETRIC-KEY ALGORITHMS PRACTICAL APPLICATIONS

José Manuel
Redondo López

- **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

Asymmetric (public key) ciphers

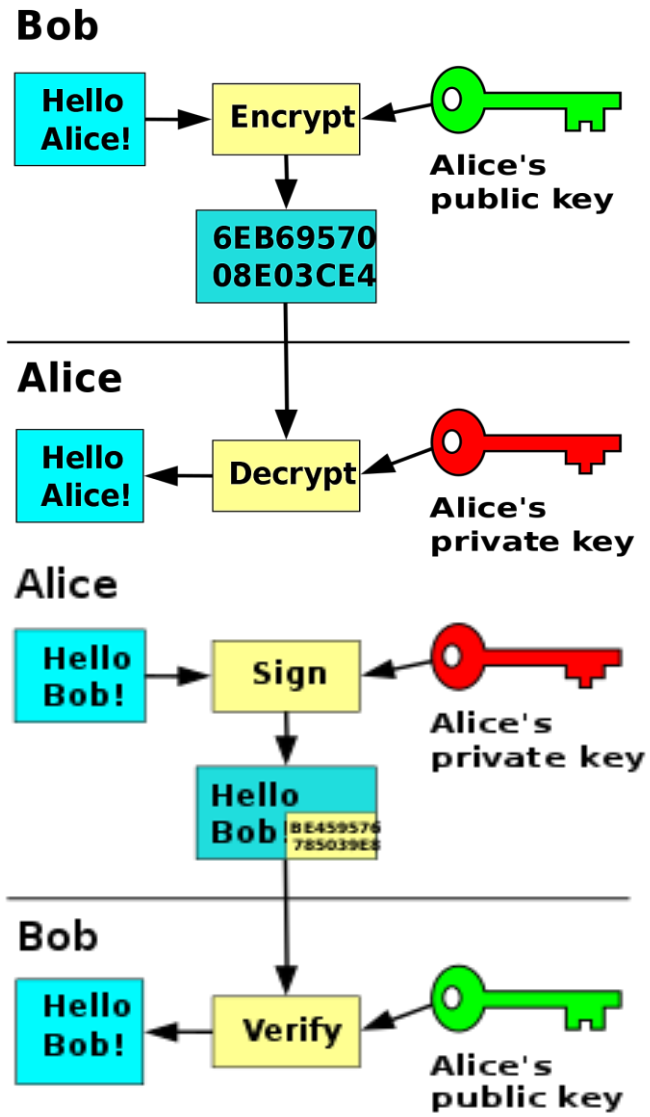
José Manuel
Redondo López

- **Avoid the key exchange problem of symmetric key ciphers**
  - Senders and receivers do not need to agree on a key, each have a **key pair**
  - **These pair of keys (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**: cryptographic methods ensure that **a particular key pair can only be generated once**
  - **The private key cannot be inferred from its corresponding public key in the pair**, so there is no danger in transmitting public keys over the network

Asymmetric (public key) algorithms

José Manuel Redondo López

- **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
    - Therefore, encrypted messages for U1 cannot be read (decrypted) by U2
    - Unless U2 steals its private key
  - Opposite, data encrypted by Alice's private key can only be decrypted by Alice's public key
    - Hence authenticating its sender (again, if the 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

RSA (Rivest, Shamir y Adleman, 1997)

José Manuel
Redondo López

- **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)**
  - 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
  - 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)

José Manuel
Redondo López

- **Public-key cryptography based on the algebraic structure of elliptic curves over finite fields**
  - https://en.wikipedia.org/wiki/Elliptic-curve_cryptography
- **Elliptic curves are applicable for key agreement, digital signatures, pseudo-random 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

José Manuel
Redondo López

- **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/

GnuPG

www.gnupg.org

José Manuel
Redondo López

- **Symmetric / asymmetric cipher and digital signing tool (next section)**
  - Successor of the PGP standard
  - V2+ also supports ECC cryptography
- **Hybrid system (symmetric + asymmetric)**
  1. Ciphers the message with a non-patented symmetric algorithm (e. g. AES)
     - Using a random key
  2. Ciphers this random key with RSA
     - Faster than ciphering all the message data
  3. The ciphered message can be sent with a cryptographic signature
- **Goals**
  - Higher encryption/decryption performance
  - Provide a form of sender authentication
- **OpenSSL also works with both types of ciphers (and hashes for integrity)**

gpg (GnuPG) 2.2.20 Cheatsheet (ingenieriainformatica.uniovi.es)

Multipurpose GPL cipher/hash/pubkey software

https://gnupg.org/

```
redondo@server1804:~$ gpg -o original.txt -d cryptandsign.enc
gpg: encrypted with 3072-bit RSA key, ID CA493341D9ED0E95, created 2019-11-18
      "redondo <redondo@mail.es>"
gpg: Signature made lun 18 nov 2019 10:12:34 UTC
gpg:                using RSA key 230C013753283601EDE49B6B4811A20BF1861B2A
gpg: Good signature from "operario <operario@mail.es>" [full]
```

```
operario@server1804:~$ gpg --list-keys
gpg: checking the trustdb
gpg: marginals needed: 3  completes needed: 1  trust model: pgp
gpg: depth: 0  valid:   1  signed:   0  trust: 0-, 0q, 0n, 0m, 0f, 1u
gpg: next trustdb check due at 2021-11-17
/home/operario/.gnupg/pubring.kbx
---------------------------------
pub   rsa3072 2019-11-18 [SC] [expires: 2021-11-17]
      230C013753283601EDE49B6B4811A20BF1861B2A
uid           [ultimate] operario <operario@mail.es>
sub   rsa3072 2019-11-18 [E] [expires: 2021-11-17]
```

GENERAL USAGE

`gpg [options] [files]`

NOTES

Sign, check, encrypt or decrypt

Default operation depends on the input data

Also supports the following compression algorithms: No compression, ZIP, ZLIB, BZIP2

OPTIONS

Symmetric encryption

Supported cipher algorithms: IDEA, 3DES, CAST5, BLOWFISH, AES, AES192, AES256, TWOFISH, CAMELLIA128, CAMELLIA192, CAMELLIA256

-c, --symmetric: encryption only with symmetric cipher

Signatures

Supported hash algorithms: SHA1, RIPEMD160, SHA256, SHA384, SHA512, SHA224

--clear-sign: make a clear text signature

--verify: verify a signature

-b, --detach-sign: make a detached signature

-s, --sign: make a signature

Asymmetric encryption

Supported public key algorithms: RSA, ELG, DSA, ECDH, ECDSA, EDDSA

-d, --decrypt: decrypt data (default)

-e, --encrypt: encrypt data

Public/private key handling (I)

--card-status: print the card status

--change-passphrase: change a passphrase

--change-pin: change a card's PIN

--check-signatures: list and check key signatures

--delete-keys: remove keys from the public keyring

--delete-secret-keys: remove keys from the secret keyring

--edit-card: change data on a card

--edit-key: sign or edit a key

--export: export keys

--fingerprint: list keys and fingerprints

Public/private key handling (II)

--full-generate-key: full featured key pair generation

--generate-key: generate a new key pair

--generate-revocation: generate a revocation certificate

--import: import/merge keys

--list-signatures: list keys and signatures

--lsign-key: sign a key locally

--print-md: print message digests

--quick-add-uid: quickly add a new user-id

--quick-generate-key: quickly generate a new key pair

--quick-lsign-key: quickly sign a key locally

--quick-revoke-uid: quickly revoke a user-id

--quick-set-expire: quickly set a new expiration date

--quick-sign-key: quickly sign a key

--receive-keys: import keys from a keyserver

--refresh-keys: update all keys from a keyserver

--search-keys: search for keys on a keyserver

--send-keys: export keys to a keyserver

--server: run in server mode

--sign-key: sign a key

--tofu-policy VALUE: set the TOFU policy for a key

--update-trustdb: update the trust database

-k, --list-keys: list keys

-K, --list-secret-keys: list secret keys

EXAMPLES

Sign and encrypt for user Bob: *gpg -se -r Bob [file]*

Make a clear text signature: *gpg --clear-sign [file]*

Make a detached signature: *gpg --detach-sign [file]*

Show keys: gpg *--list-keys [names]*

Show fingerprints: *gpg --fingerprint [names]*

Cipher a file using a strong symmetric key algorithm: *gpg --symmetric --cipher-algo AES256 –c message.txt*

Cipher files with a user-friendly Data Format: *gpg --armor --symmetric --cipher-algo AES256 –c message.txt*

Decipher a file that used symmetric encryption: *gpg --decrypt --output=dmessage.txt message.txt.gpg*

Clear signature for a file: *gpg --clearsign file.txt*

Assymetric encryption for a particular user (redondo): *gpg -o file_for_redondo.gpg -e -r redondo file.txt*

Decryption of asymmetrically encrypted text: *gpg -o file.txt -d file.txt.gpg*

Miscellaneous options

--openpgp: use strict OpenPGP behavior

--textmode: use canonical text mode

-a, --armor: create ascii armored output

-i, --interactive: prompt before overwriting

-n, --dry-run: do not make any changes

-o, --output FILE: write output to FILE

-r, --recipient USER-ID: encrypt for USER-ID

-u, --local-user USER-ID: use USER-ID to sign or decrypt

-v, --verbose: verbose

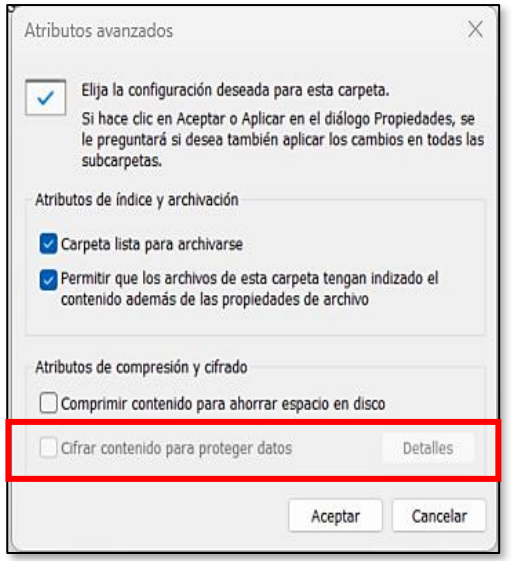
-z N: set compress level to N (0 disables)

ASYMMETRIC ALGORITHMS: PRACTICAL APPLICATIONS

José Manuel
Redondo López

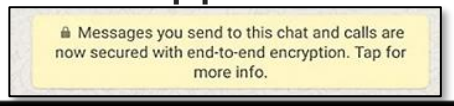
File and directory encryption

- **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
  - Single general-purpose file systems with 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
  - Cryptographic keys are only held in memory while its associated file is open



End-to-end encryption

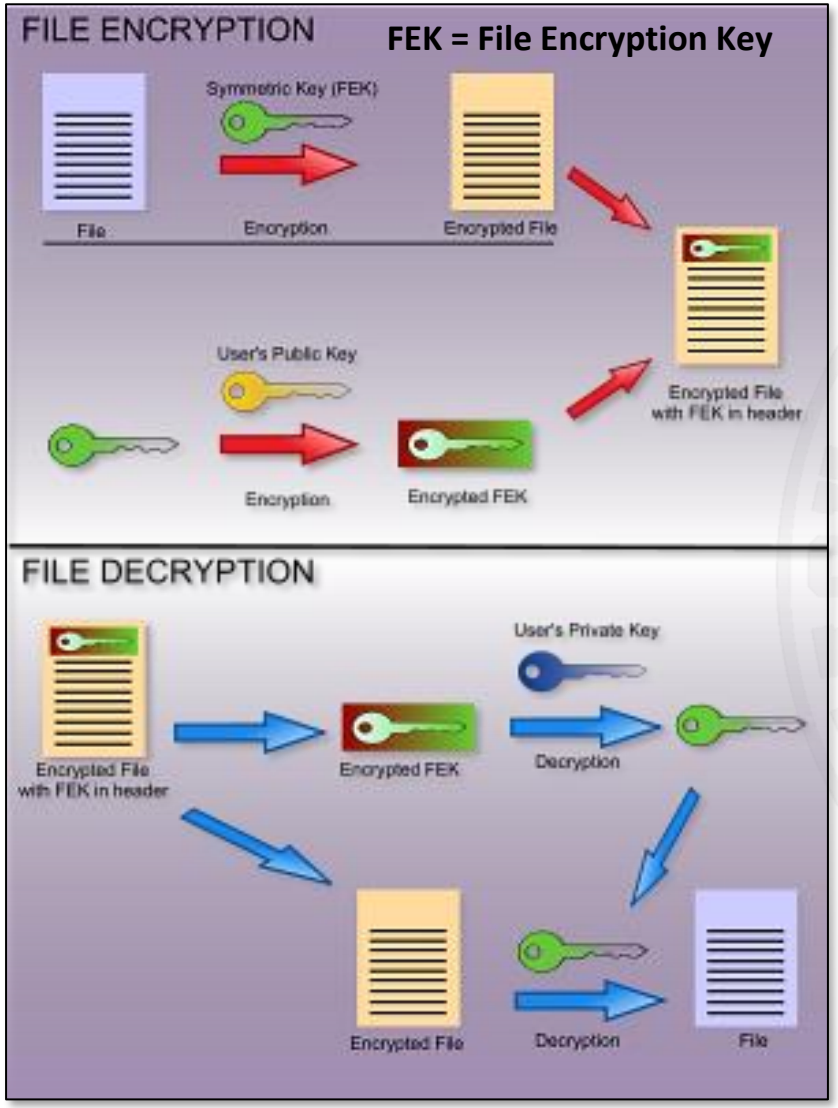
- Secure communications between pairs of users, typical of lots of messaging and cloud applications
- WhatsApp, Telegram…

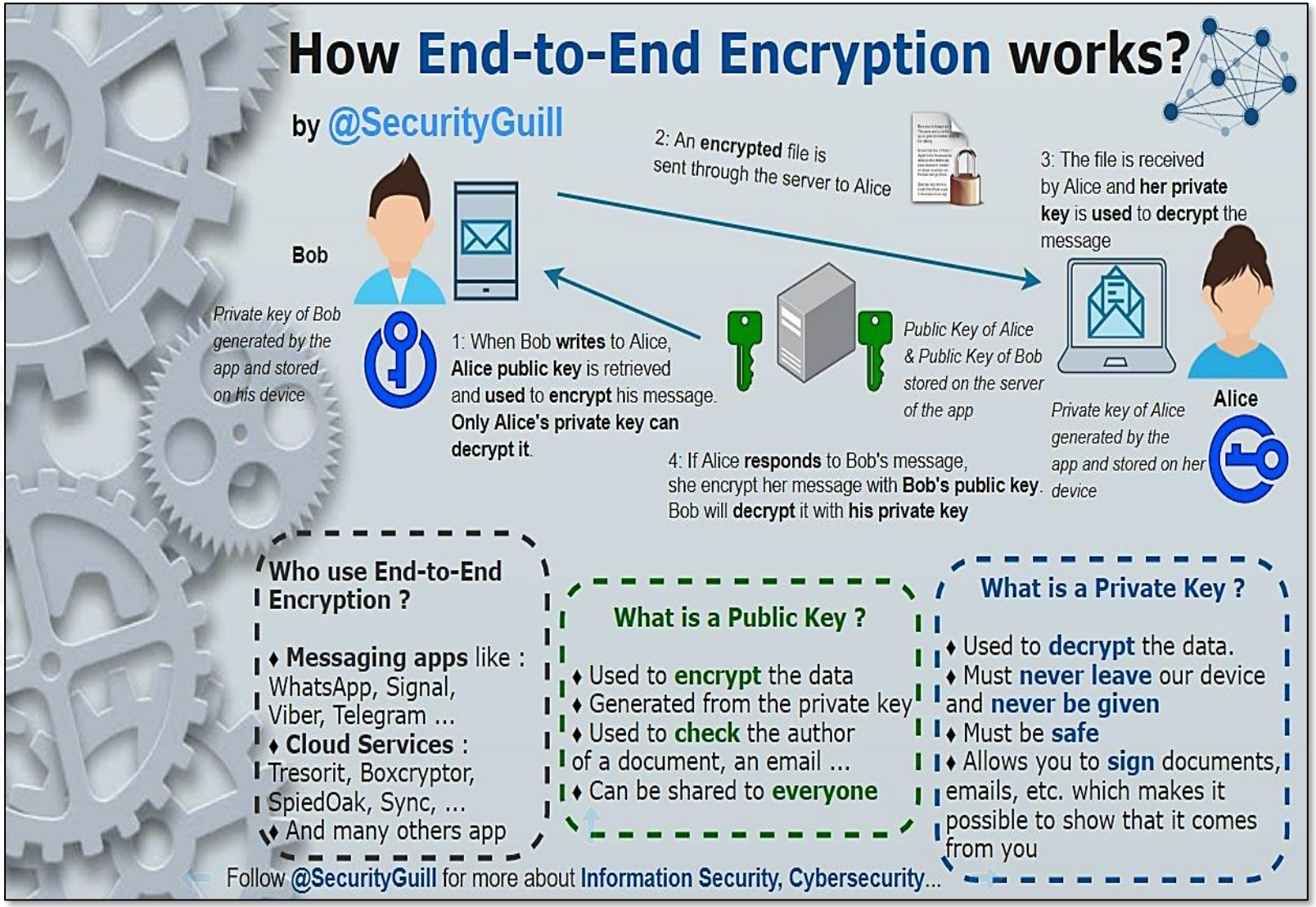


Source: WhatsApp

ASYMMETRIC ALGORITHMS: PRACTICAL APPLICATIONS

José Manuel
Redondo López





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

< Go to Index

Source: zona-militar.com

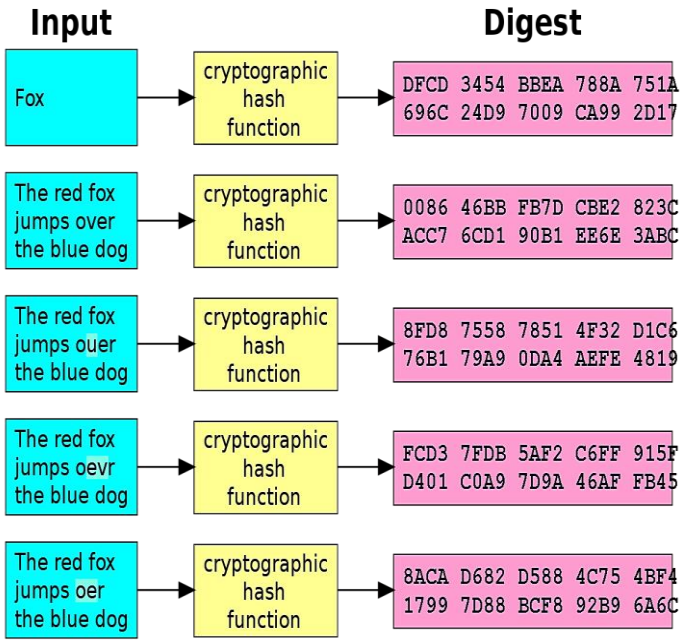
Integrity

Is this content unaltered?

HASH OR SUMMARY FUNCTION

José Manuel
Redondo López

- **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 Hash(M), 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 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

José Manuel
Redondo López

- **First preimage resistance**
  - If you know a hash value, 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-length) and **SHA-1** (Secure Hash Algorithm, 160-bit length)
  - **Obsolete and must NOT be used**
- SHA-2 algorithms are recommended 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

José Manuel
Redondo López

- **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

```
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

José Manuel
Redondo López

- **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 an calculate 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)

José Manuel
Redondo López

- **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)
  - **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!

Password storage: Key Derivation Function (KDF)

José Manuel
Redondo López

- **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)
```

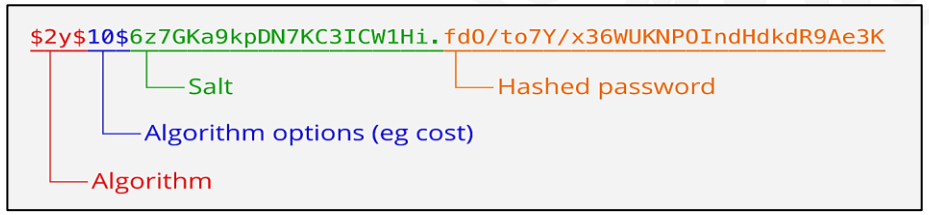
Password storage: Key Derivation Function (KDF)

José Manuel
Redondo López

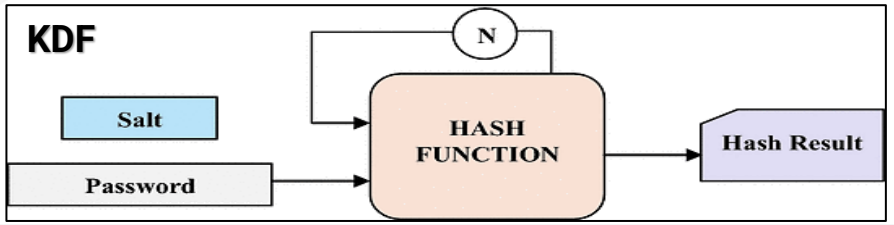
- **Resistance to these attack types depend on the key length and the amount of computational entropy that could be generated while generating the DK**
  - The more, the better security
  - This is the randomness collected by an OS / application to use in cryptography, among other uses
- **Known modern 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
  - **Argon2**: It has three versions specifically parametrized against different attacks
    - Winner of the Password Hashing Competition in July 2015; The recommended version is Argon2id

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

José Manuel
Redondo López

- **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 one
    - 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

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

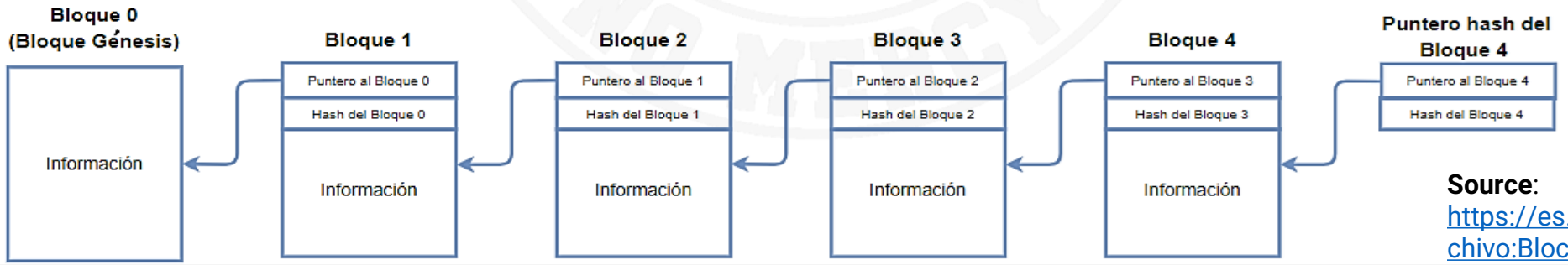
Note: By definition, it takes half of the *guaranteed* crack time on *average* to crack a password

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)

BLOCKCHAIN

José Manuel
Redondo López

- **The first block (block 0) of the chain is known as the genesis block**
  - Each other block stores a hash 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 one: 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 then it 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

< Go to Index

Source: zona-militar.com

Digital sign >

Auth. Encryption >

Certificates >

AUTHENTICATION

Who do I think you are?

Digital Signatures

Not the photo of your physical signature! ☺

DIGITAL SIGNATURE

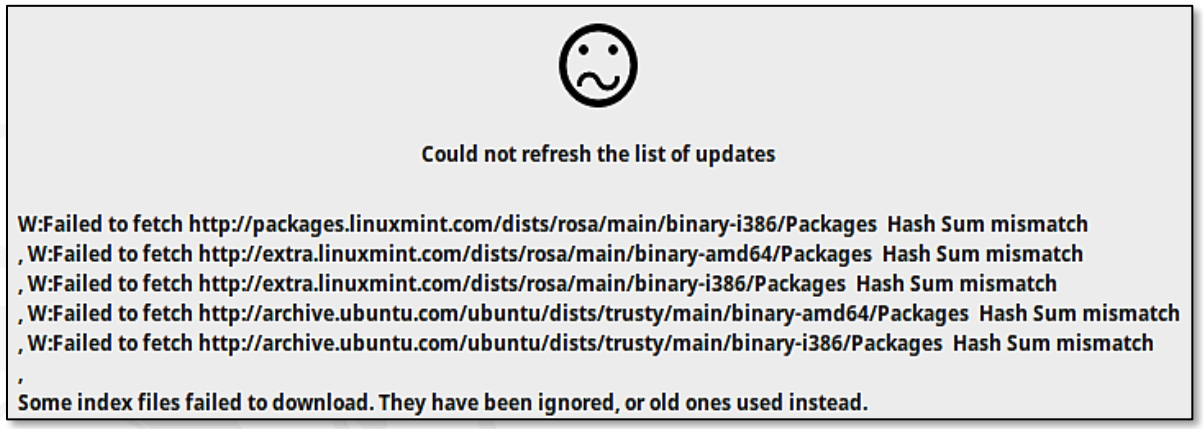
José Manuel
Redondo López

- **Result of a cryptographic operation that joins a signed document to a personal component, 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
- **The signed file can be verified**
  - The original file hash is obtained with the user public key from the signed one
  - If the document was modified in any way, this hash will be different from the hash calculated from the current 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**

Digital signature. Objectives

José Manuel
Redondo López

- **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**



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

Digital signature. Properties

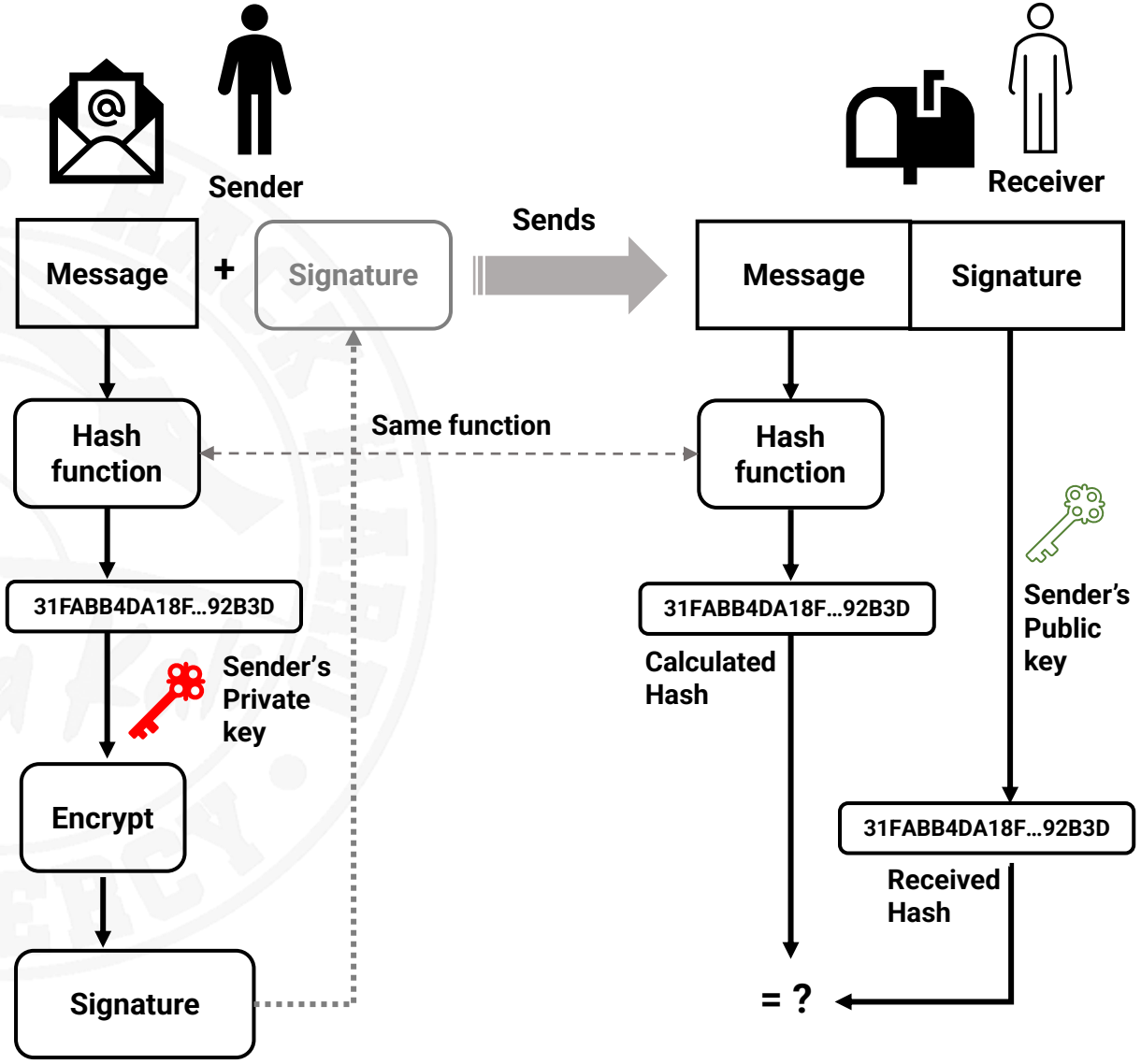
José Manuel
Redondo López

- **To ensure its objectives, a signature must have these properties**
  - It must be **unfalsifiable**
  - It must **not be reusable**
    - It must be part of the document, and impossible to move to any other document
  - The document **must not be altered once signed**
- **Nowadays, digital certificates are the preferred way of ensuring these properties**
  - They are the most typical way of associating **a public key with the machine or person** that sends a message
  - The public key is necessary to verify a signature of the user we received
  - Digital certificates will be reviewed later

Digital signature & Hash functions: Message signing procedure

José Manuel
Redondo López

- **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!



Asymmetric ("public key") signatures

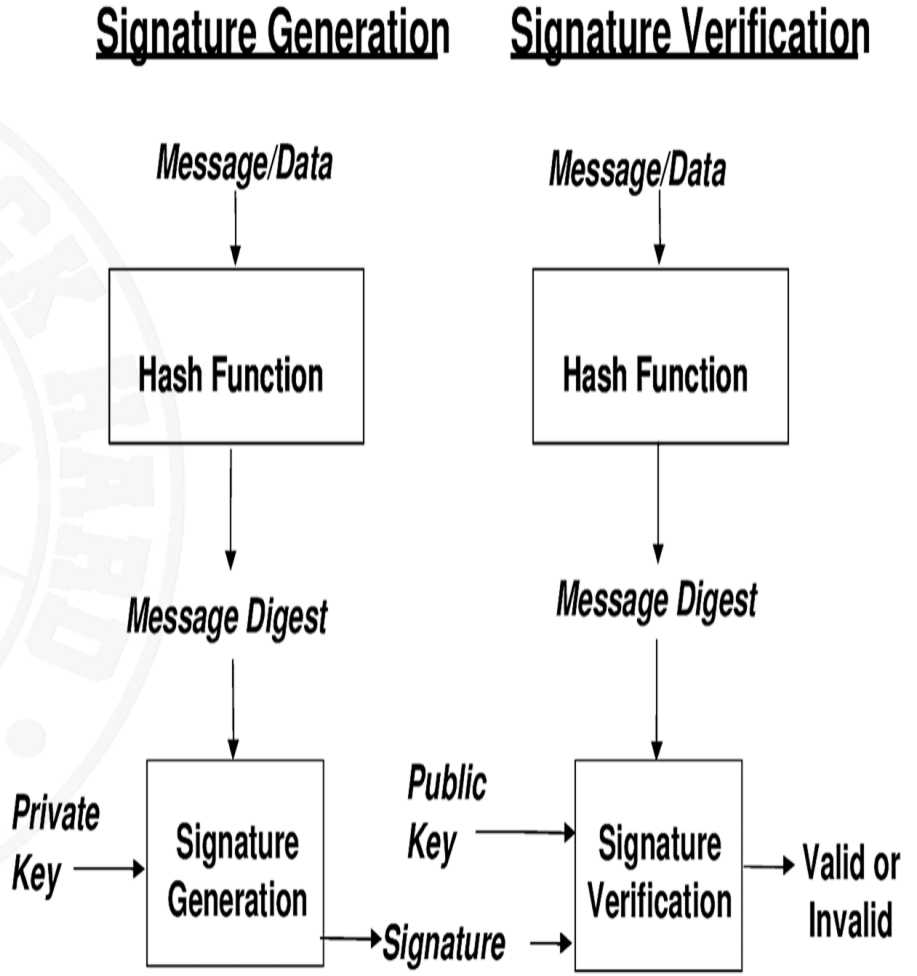
José Manuel
Redondo López

- **Use the following algorithms, in order of preference**
  - **Ed25519** (fixed key size): Fast and highly secure public-key signature system
  - **ECDSA** with the **secp256r1** curve (fixed key size)
    - The Elliptic Curve Digital Signature Algorithm (**ECDSA**) is variant of the Digital Signature Algorithm (**DSA**) that uses elliptic curve cryptography
  - **RSA** with (at least) 2048-bit keys
- **All we said about RSA encryption applies to RSA signatures**
  - ECDSA: 256-bit keys (much sorter and computational less intensive to calculate!)
  - RSA: 2048-bit keys
- **There are different ways to create digital signatures**
  - **DSS** (Digital Signature Standard)
  - **RSA-PSS**, **PKCS#1**, **PGP**, **GnuPG** (that can also provide authenticated encryption)…

DSS (Digital Signature Standard)

José Manuel
Redondo López

- **Allows document signing, but no encryption**
- **It follows the described procedure**
  - Uses a public key algorithm (DSA, RSA, ECDSA…) to sign data
  - Uses a hash function (e. g. SHA-2) over the data
    - This hash is encrypted with the private key of the sender
  - Message + its encrypted hash are transmitted together
  - When the message reaches its destination, the receiver uses the sender's public key to decrypt the hash
    - It applies the same hash function to the data, calculating a hash locally
    - If the local hash is the same as the one that the deciphered one brought along the message, we can ensure message authenticity and integrity

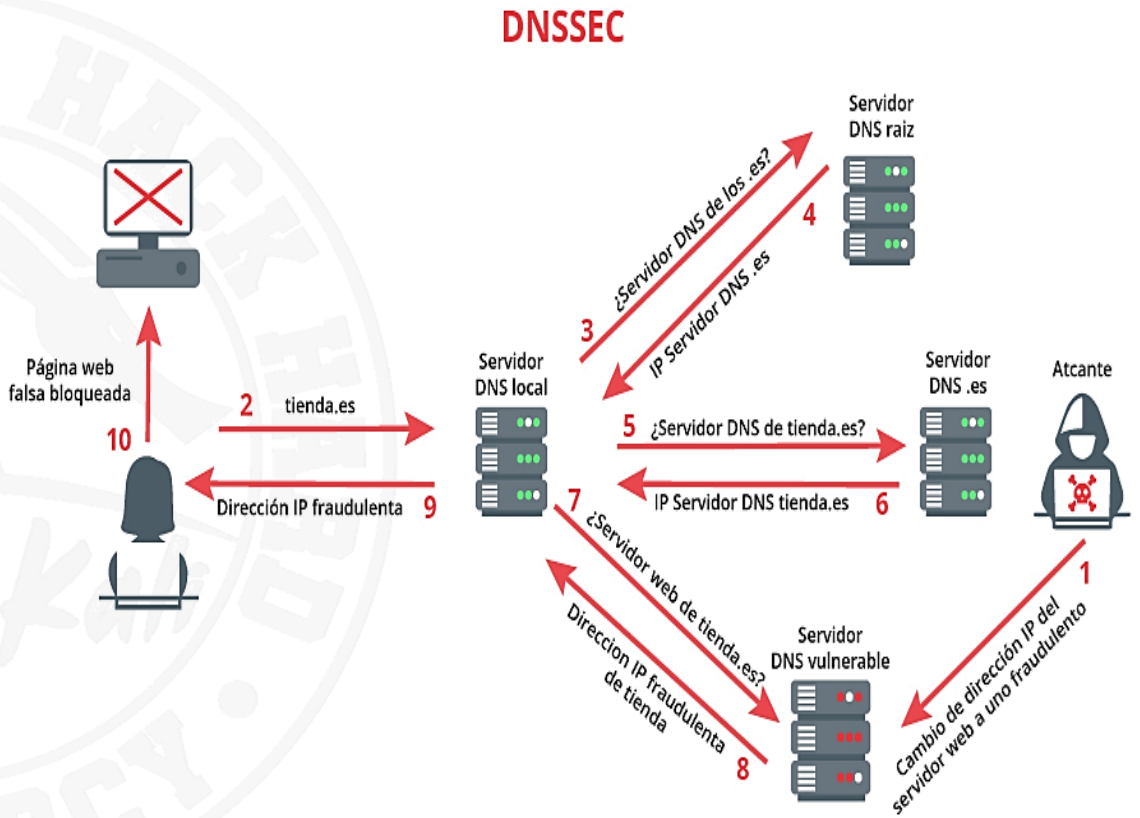


Source: https://www.semanticscholar.org/paper/Digital-Signature-Standard-(DSS)-Fox/9dc716a7192de932e80016c491091d770de01a12/figure/3

APPLICATIONS OF DIGITAL SIGNATURES: DNSSEC

José Manuel
Redondo López

- **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

José Manuel
Redondo López

- **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**

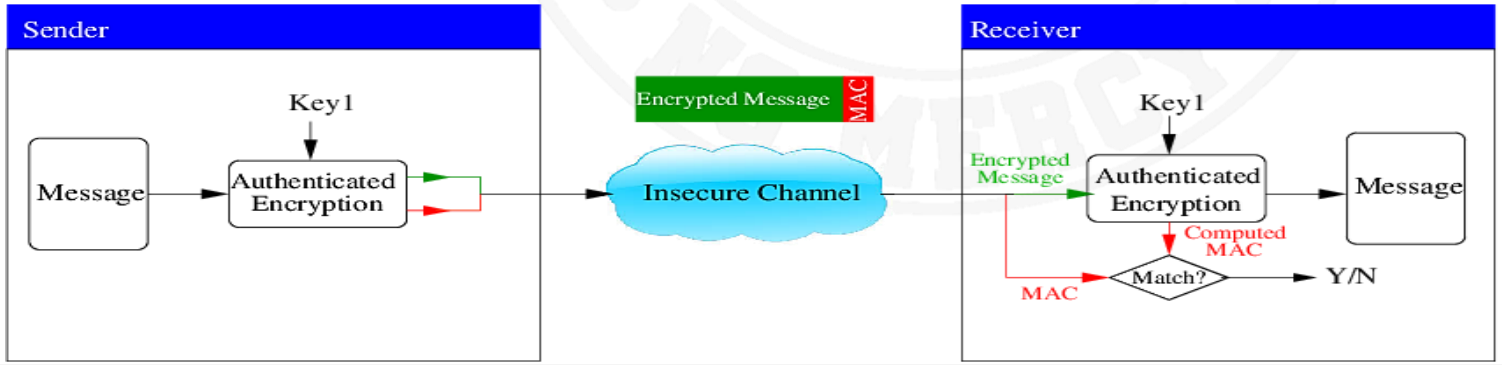
Authenticated Encryption

MACs without Apples ☺

MAC (Message Authentication Code)

José Manuel
Redondo López

- **Symmetric-key algorithms that just encrypt data are not enough**
  - Message integrity must also be ensured
  - Authenticated encryption is the **preferred method** to guarantee **Integrity**, **Authentication,** and **Confidentiality**
  - **RSA** is even discouraged in some contexts (e. g. TLS 1.3) to achieve these three properties
- **MAC is a procedure that adds to a message a code that depends on a shared symmetric key, authenticating the message and the sender**
  - MAC codes are normally **small and have a fixed length**
  - Previous schemas combined separate confidentiality and authentication systems, MAC does all by itself
- **There are 3 ways of creating MACs**
  - **Hash-based** (HMAC, **recommended**), **Block cipher-based** (CBC-MAC), and **Galois Counter Message Authentication Code-based** (GMAC)

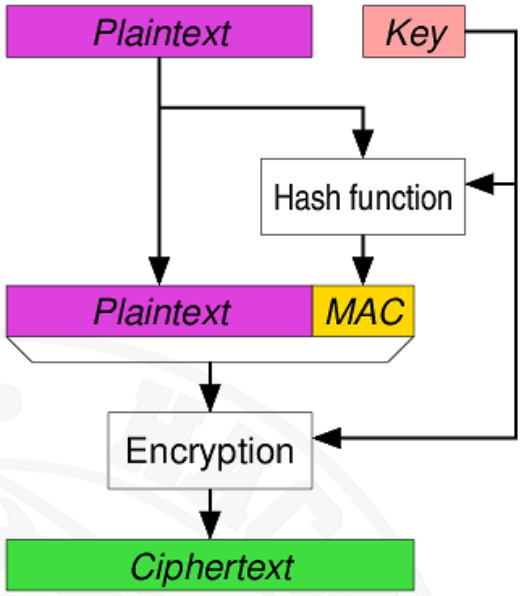


Source:
https://www.researchgate.net/publication/279264224_Authenticated_Encryption_on_FPGAs_from_the_Reconfigurable_Part_to_the_Static_Part/figures?lo=1

MAC CALCULATION

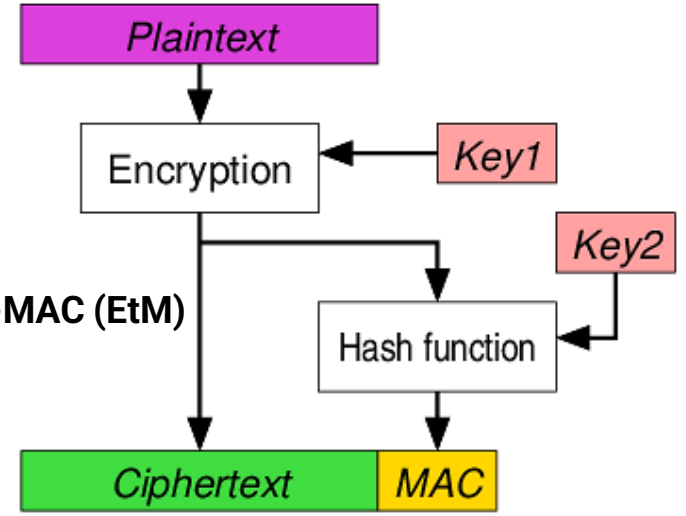
José Manuel
Redondo López

- **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

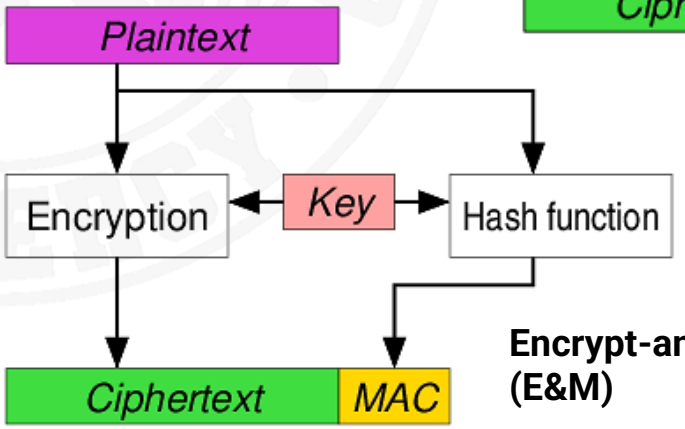


Source:
https://en.wikipedia.org/wiki/Authenticated_encryption

Mac-then-Encrypt (MtE)



Encrypt-then-MAC (EtM)



Encrypt-and-MAC (E&M)

AUTHENTICATED ENCRYPTION IMPLEMENTATIONS

José Manuel
Redondo López

- **Authenticated Encryption (AE) and Authenticated Encryption With Associated Data (AEAD) are ciphering forms that provide the three enunciated properties (CIA)**
- **Some popular implementations are**
  - **XChaCha20-Poly1305, XSalsa20-Poly1305**
    - Salsa20 and Chacha20 are closely related symmetric stream ciphers
      - ChaCha is a modification of Salsa that increases diffusion and increases performance on some architectures
      - Variants of both algorithms with 192-bit nonces have names that begin with "X"
    - Poly1305 is a popular **cryptographic message authentication code** (CBC-MAC)
      - Google selected Poly1305 combined with the ChaCha20 symmetric cipher as a replacement for RC4 in TLS/SSL
      - Providing authentication, integrity and encryption with this combination
      - □λ□□λ□□λ□□ε□□ε□□ε□□ε□□ε□□ε□□λ□: an attacker must break AES to break Poly1305 (strong cipher)
  - **SipHash** is another similar famous algorithm to calculate MACs
    - https://en.wikipedia.org/wiki/SipHash
  - **GnuPG** also supports authenticated encryption modes
    - https://gnupg.org/documentation/manuals/gcrypt/Available-cipher-modes.html

AUTHENTICATED ENCRYPTION IMPLEMENTATIONS

José Manuel
Redondo López

- **To ensure CIA, the most recommended methods are, in order**
  - XChaCha20-Poly1305 or XSalsa20-Poly1305 (which always have 256-bit keys)
  - AES-GCM-SIV (regardless of key size)
  - ChaCha20-Poly1305 (which always has 256-bit keys)
  - AES-GCM (regardless of key size)
- **If using a reputable TLS library (like OpenSSL), all options are adequate**
- **For application-layer symmetric-key encryption, authentication, and integrity two additional options should be considered**
  - AES-CTR (regardless of key size) + HMAC-SHA2 (EtM MAC generation method)
  - AES-CBC (regardless of key size) + HMAC-SHA2 (EtM MAC generation method)
  - 128-bit or 256-bit keys are both fine with these two options

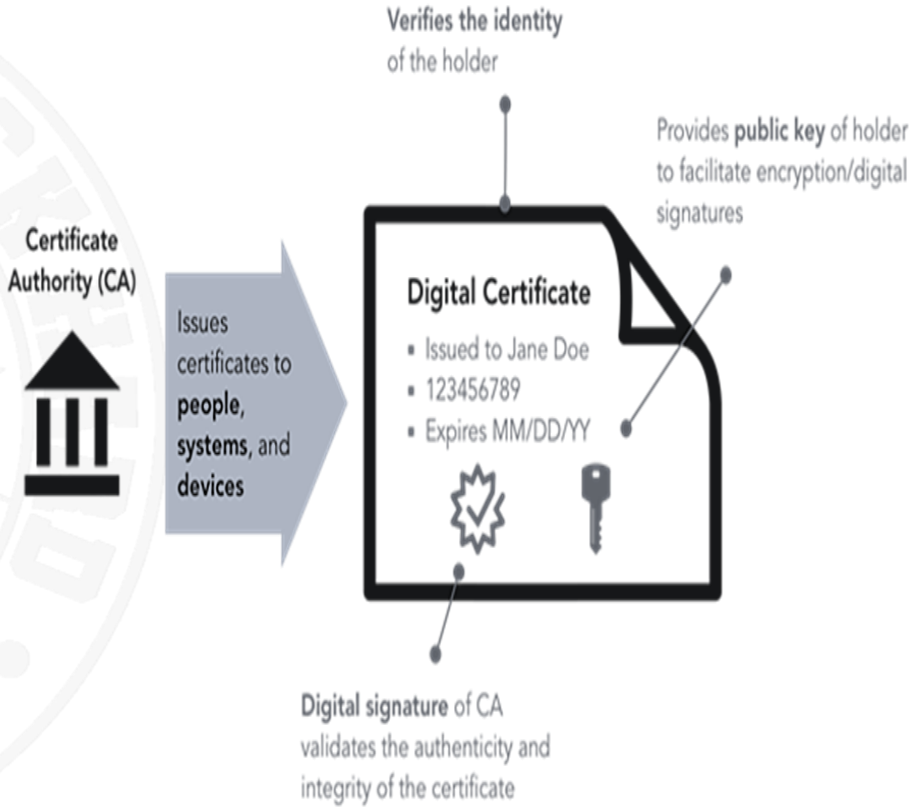
Digital Certificates

A digital document to identify them all

DIGITAL CERTIFICATES

José Manuel
Redondo López

- **They are used to authenticate public keys**
  - Asymmetric cryptography cannot work if we cannot ensure that a certain public key belong to an 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 public key,**
  - And validated by a trusted entity (E. g.: CA)
- **How is the trust of the association modeled?**
  - Ensure non-fake identities
  - Provide non-corrupt keys



Source: https://id4d.worldbank.org/guide/digital-certificates-and-pki

User key authentication

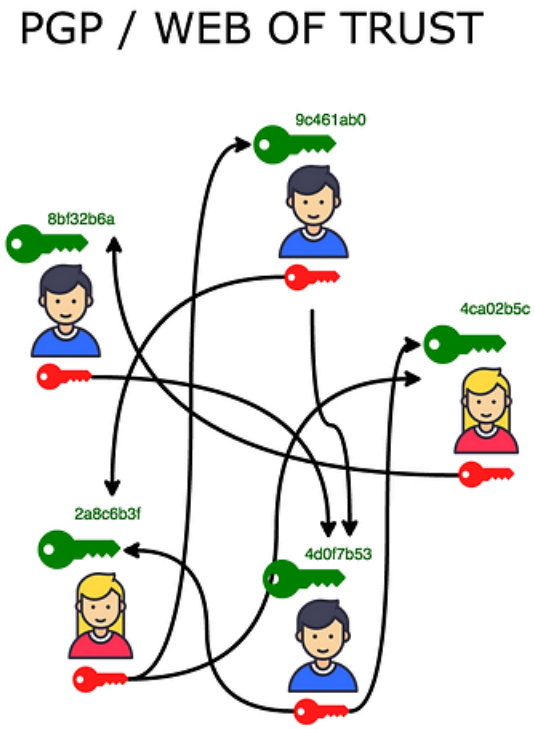
José Manuel
Redondo López

- **We need an additional mechanism to make sure that the holder of a key is who it claims to be**
- **Nowadays, these mechanisms follow a similar 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 model of trust
  - **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 the associated certificates

Trusted rings (GnuPG)

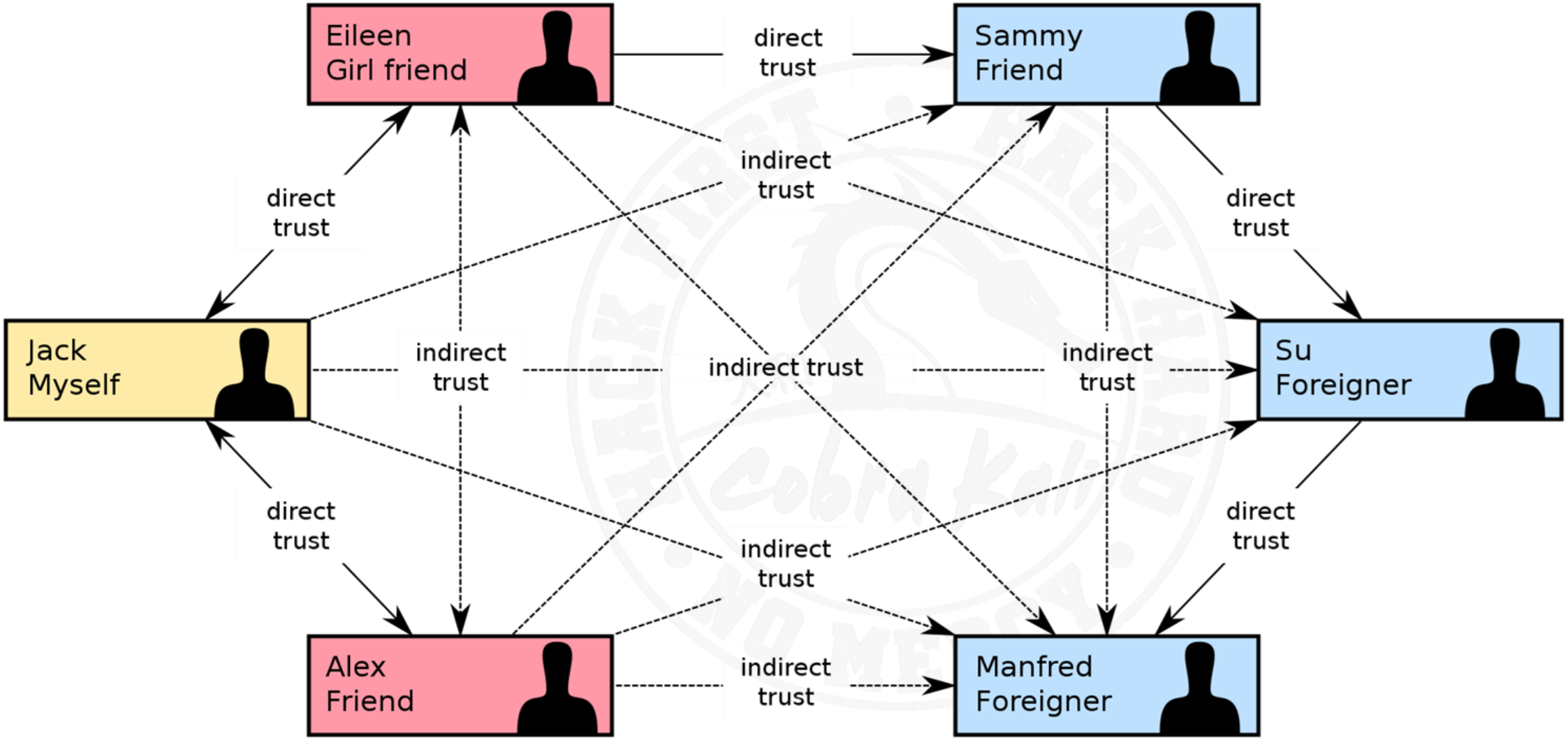
José Manuel
Redondo López

- **When we get a public key from a user, we can associate to it a level of trust**
  - **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
- **When we get another user's public key…**
  - It 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 (and image source):** https://cran.r-project.org/web/packages/gpg/vignettes/intro.html



Trusted ring (Web of trust)

José Manuel
Redondo López



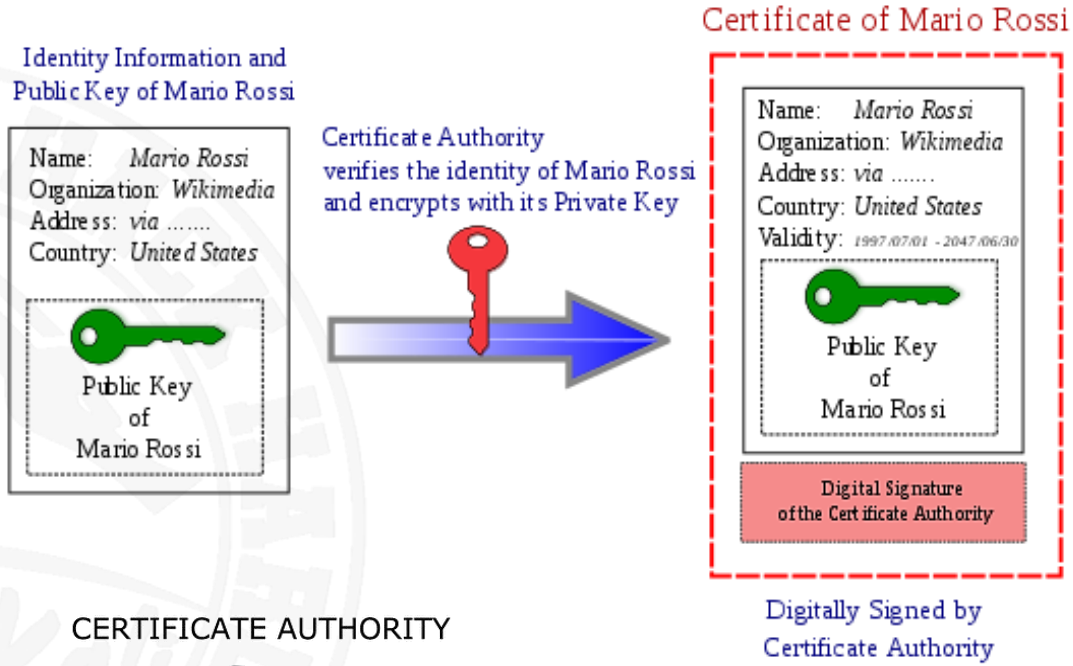
https://en.wikipedia.org/wiki/Web_of_trust#/media/File:Web_of_Trust-en.svg

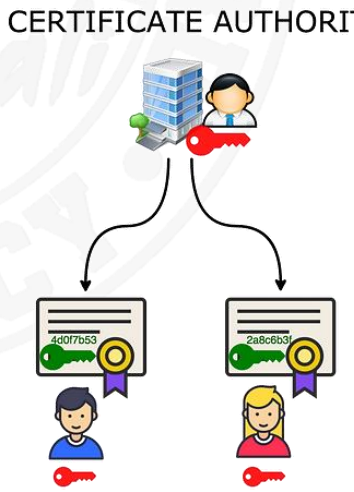
Certification Authority (CA)

José Manuel
Redondo López

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

- **They sign their certificates with its own private key to ensure its authenticity**
  - It acts like a notary, certifying the validity of the certificate
- **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 integrity





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

Certification Authority (CA)

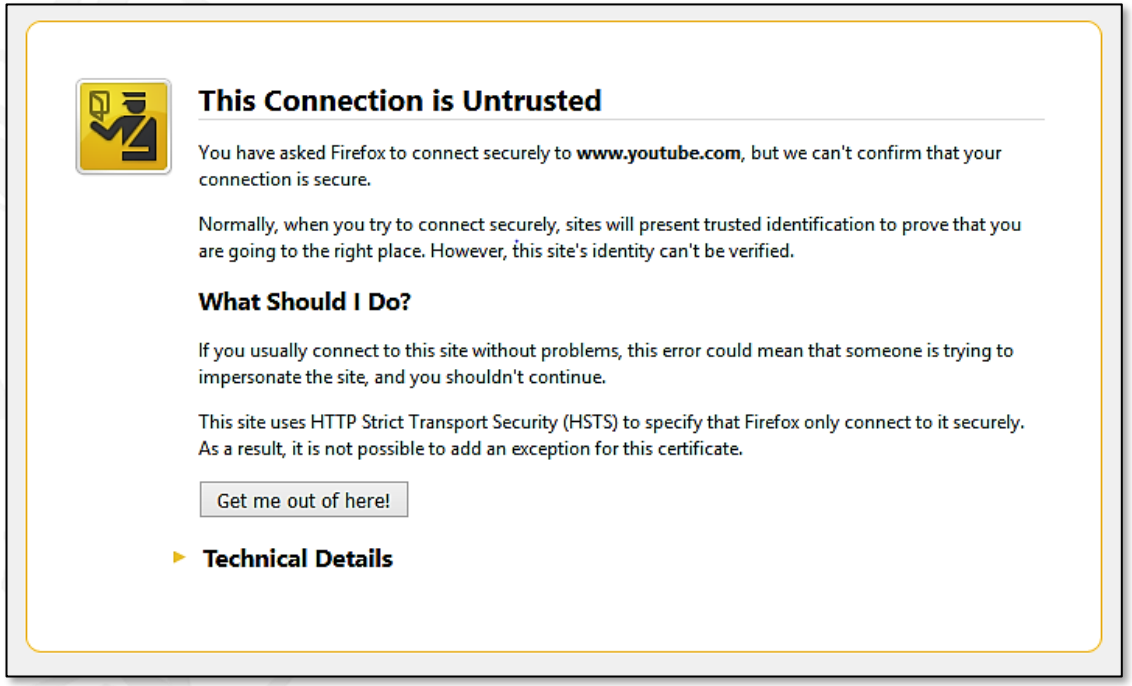
José Manuel
Redondo López

- **In this trust scheme, 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 an open 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

José Manuel
Redondo López

- **If we use non-trusted CAs (like our own ones), we obtain warnings or errors**
  - Browsers include their own list of trusted CAs
  - Using one not in this list gives us a warning
  - Normally this happens when implementing HTTPs with a self-signed certificate
  - **Typical in testing environments, 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

José Manuel
Redondo López

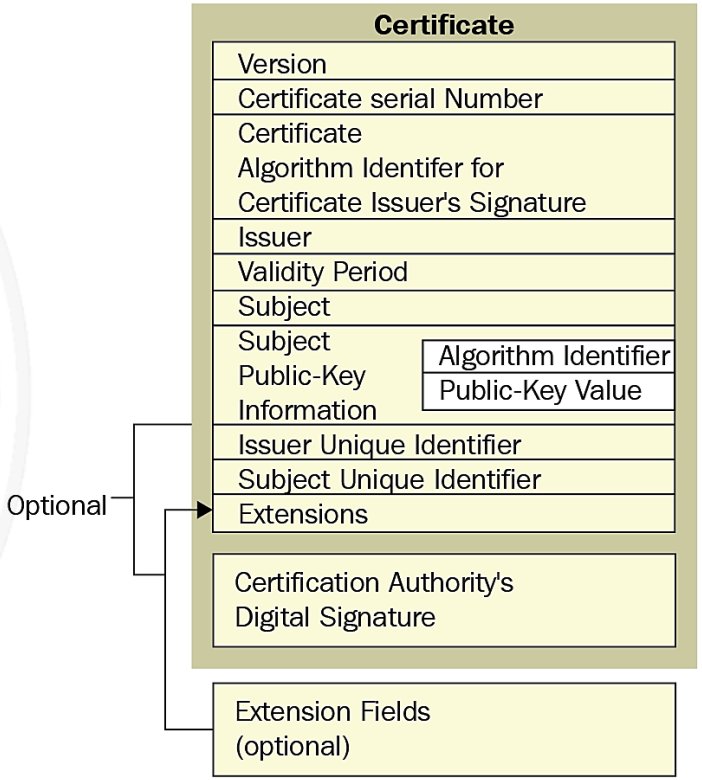
- **Personal**: Issued to users for tasks like authentication or digital signatures
  - A single user may have multiple ones, each one signed by the CA that issued it
  - Personal certificates usually may have additional uses
    - **Company membership**: In addition to person identity, certifies that 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 software 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 entities
  - "(Luke) I am your CA"

DIGITAL CERTIFICATES: STRUCTURE

José Manuel
Redondo López

- **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**

Standard X.509 certificate format



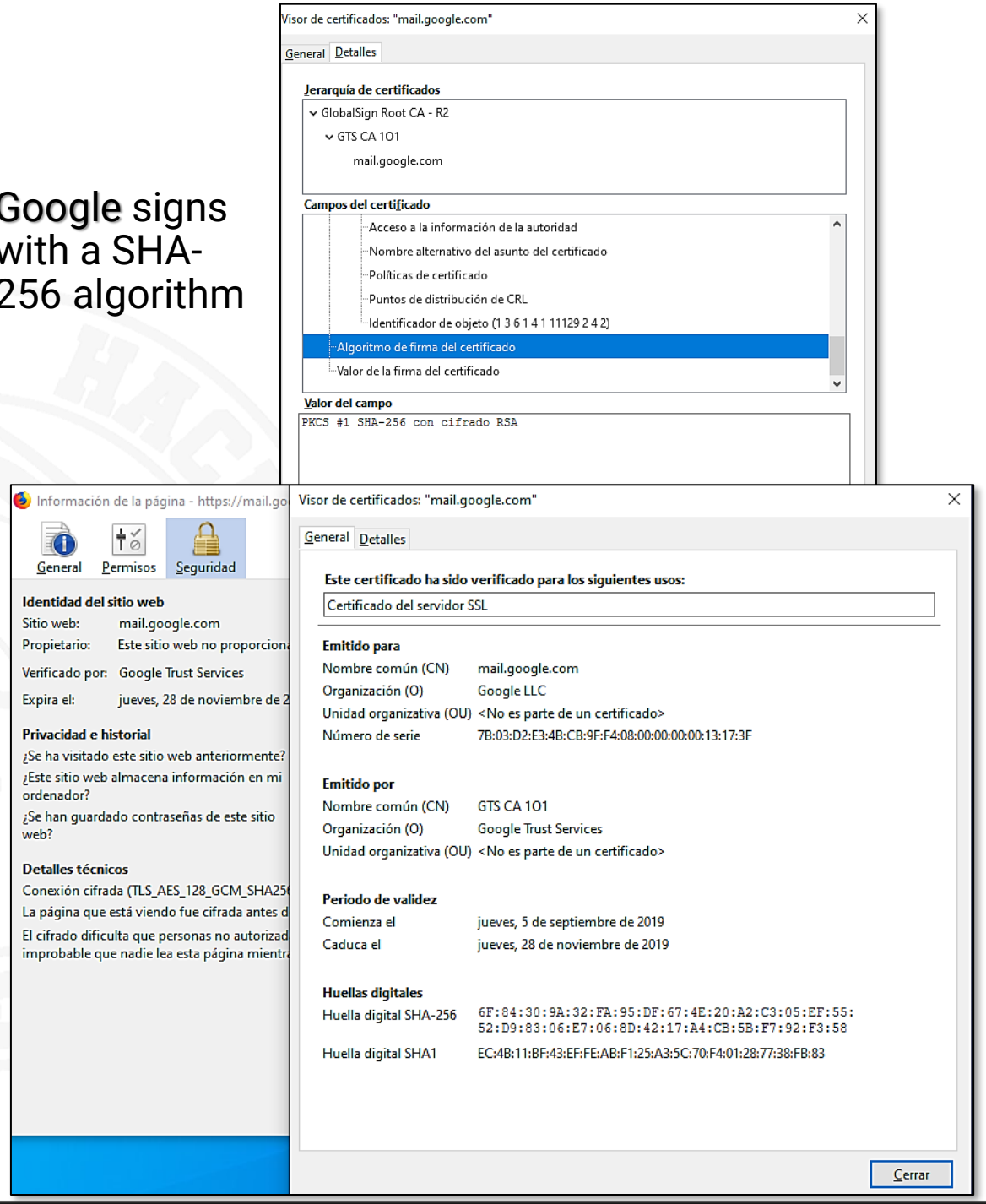
Source:
https://subscription.packtpub.com/book/business/9781788832687/3/ch03lvl1sec31/pki-certificate-standards-for-iiot

IDENTIFYING SERVERS

José Manuel
Redondo López

- **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**!

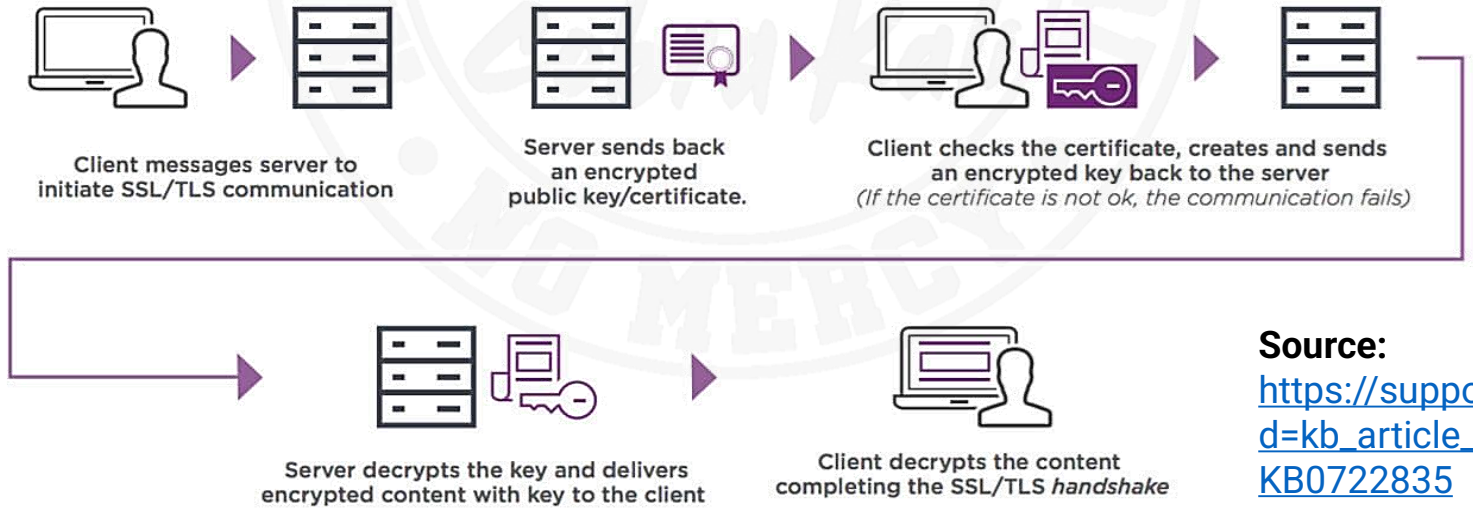
Google signs with a SHA-256 algorithm



REMOTE TRANSACTIONS: HTTPS

José Manuel
Redondo López

- **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 with **old clients** (Windows XP)
  - Or **misconfigured web servers** (cryptographic downgrade attacks)
- **Configuring any webserver with adequate HTTPs protocol features is easy following this guide: https://ssl-config.mozilla.org/**

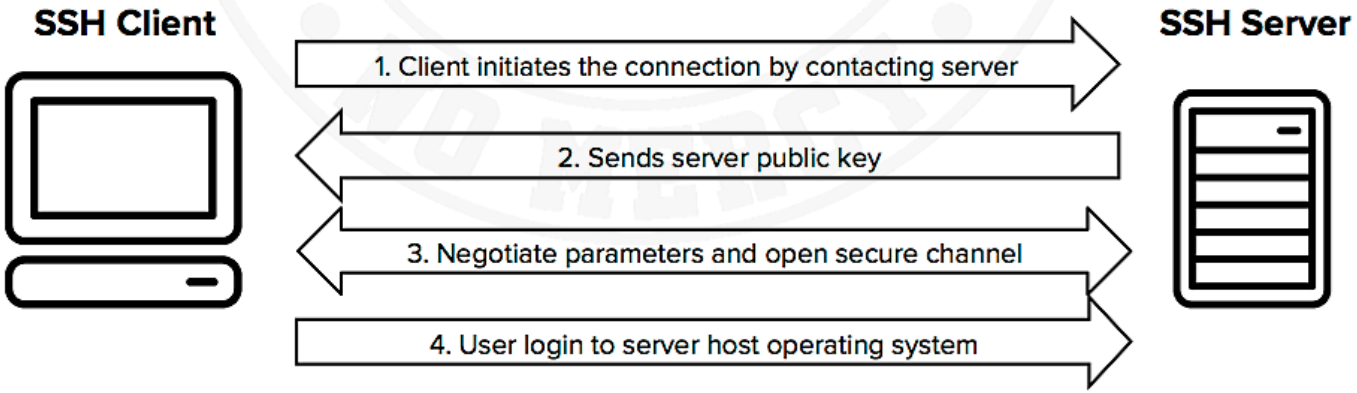


Source:
https://support.servicenow.com/kb?id=kb_article_view&sysparm_article=KB0722835

REMOTE TRANSACTIONS: SSH

José Manuel
Redondo López

- **You can use SSH to encrypt connections to a remote machine**
  - At the beginning of the connection the server sends its public key to the client
  - If the client accepts this key, it generates a random session key, encrypts it with the server key, and sends it back
  - This key will be used for encrypted transmission of the data
    - With a symmetric algorithm
    - 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

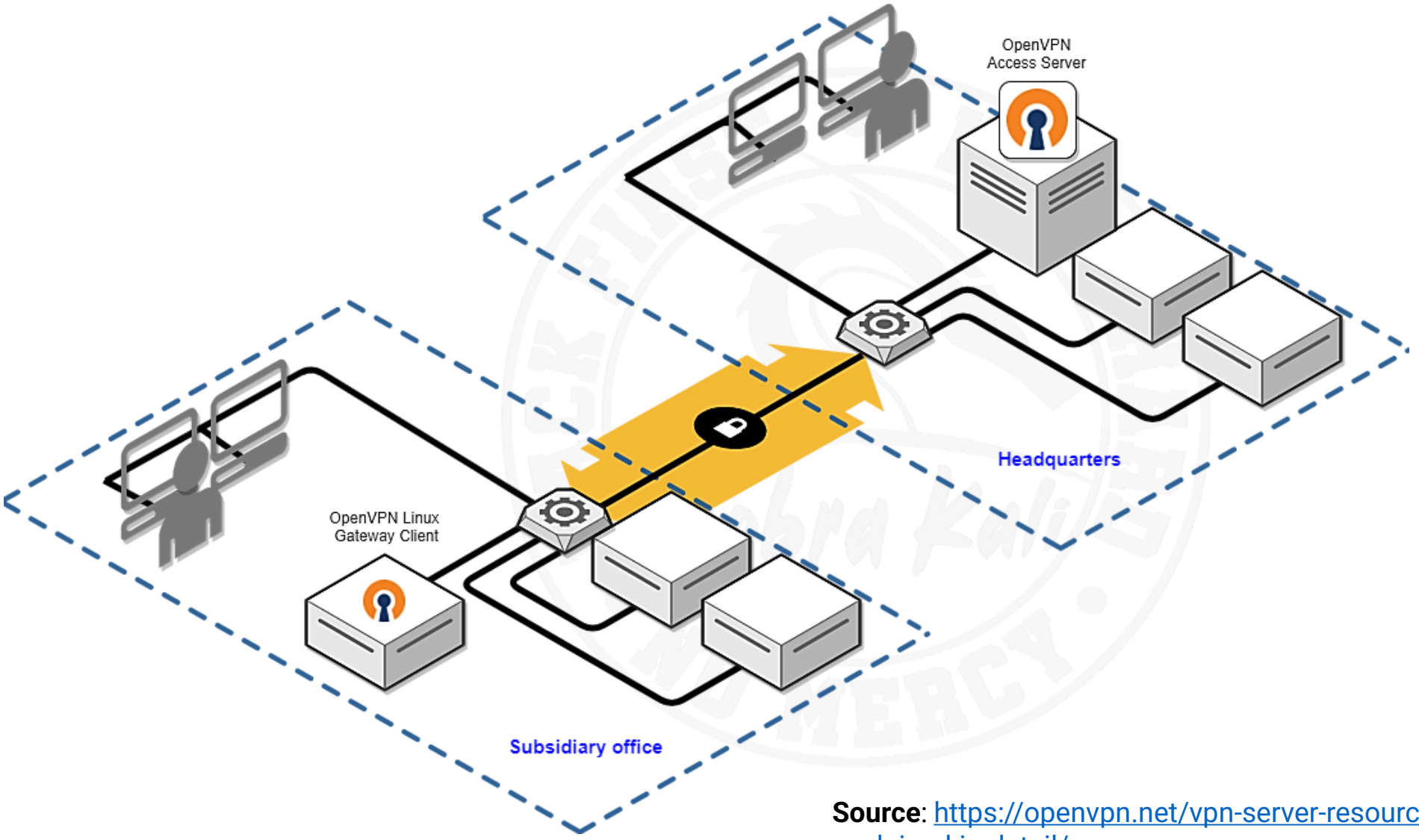
Virtual Private Networks (VPNs)

José Manuel
Redondo López

- **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 computing 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/
  - OpenVPN is also popular, but with a weaker default security
    - This guide may strengthen it: https://blog.g3rt.nl/openvpn-security-tips.html
  - ExpressVPN: Is the TechRadar 2020 and CNET Editor's Choice: www.expressvpn.com

Virtual Private Networks (VPNs)

José Manuel
Redondo López

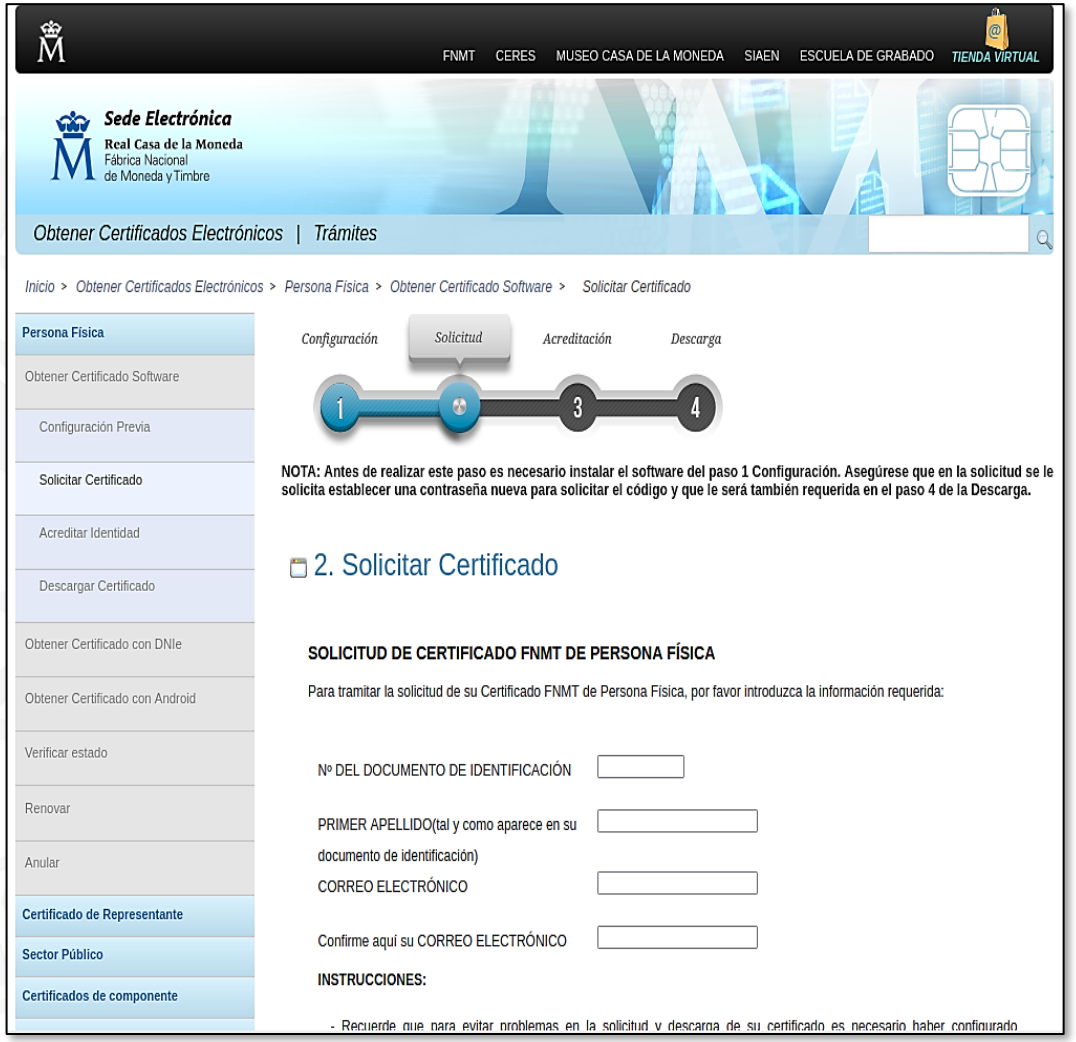


Source: https://openvpn.net/vpn-server-resources/site-to-site-routing-explained-in-detail/

REMOTE TRANSACTIONS: CUSTOMERS / CLIENTS / CITIZENS

José Manuel
Redondo López

- **Certain administrative procedures also 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**
- **Official institutions often recognize certificates issued by the Ceres**
  - From the National Mint and Stamp Factory
  - http://www.cert.fnmt.es/

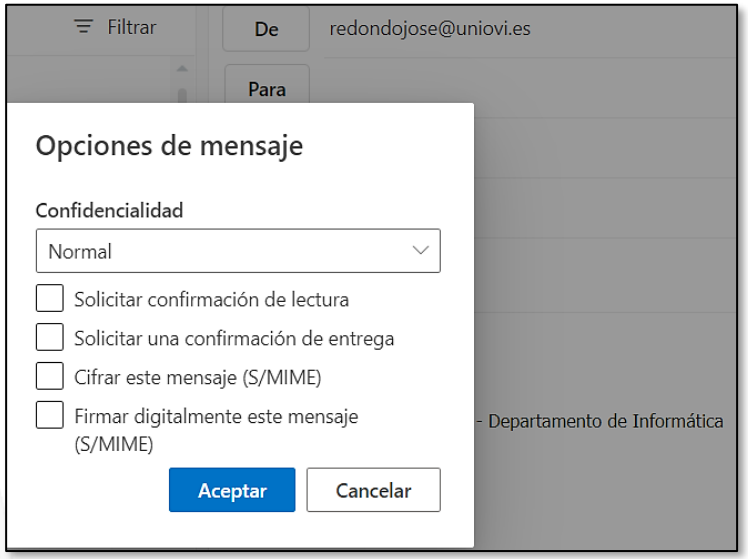


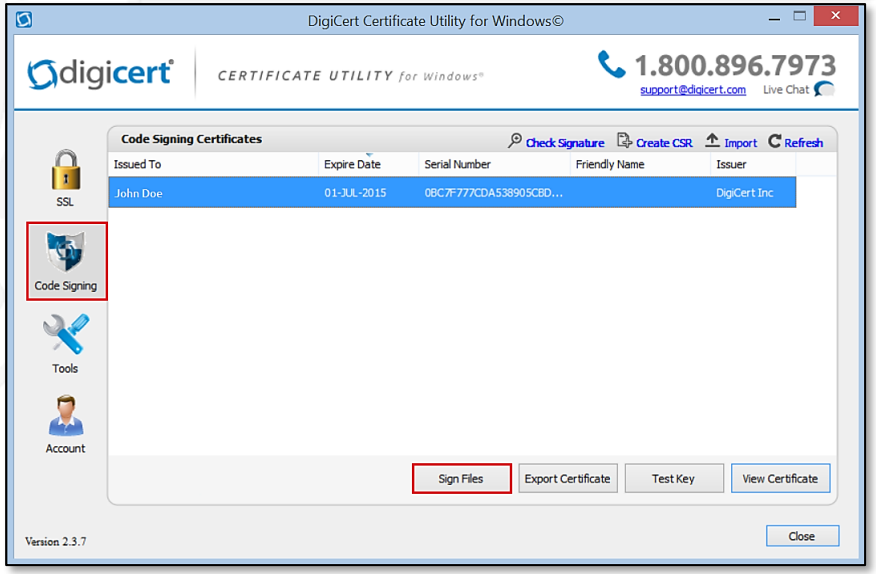
Source: https://www.sede.fnmt.gob.es/certificados/persona-fisica/obtener-certificado-software/solicitar-certificado

EMAIL / FILE AUTHENTICATION

José Manuel
Redondo López

- **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
  - Also, certain webmail services (e. g. **Hushmail, 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**...)



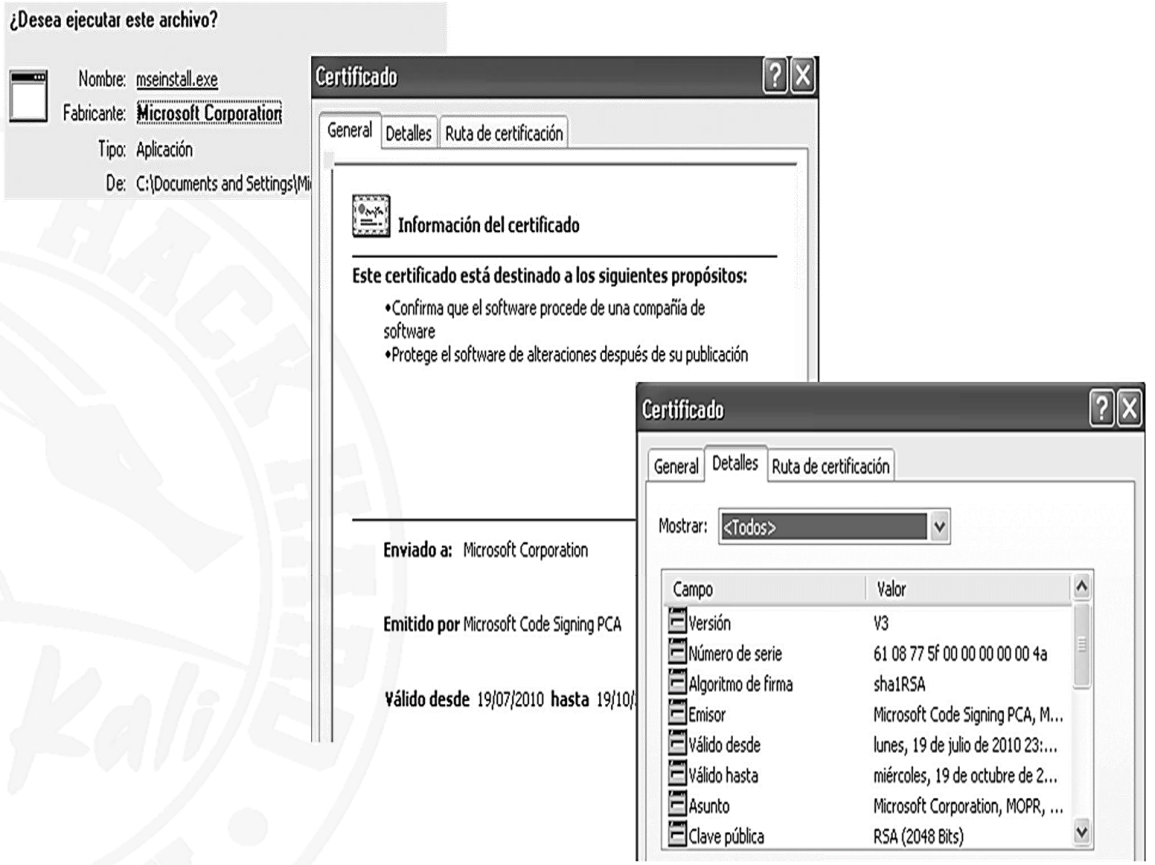


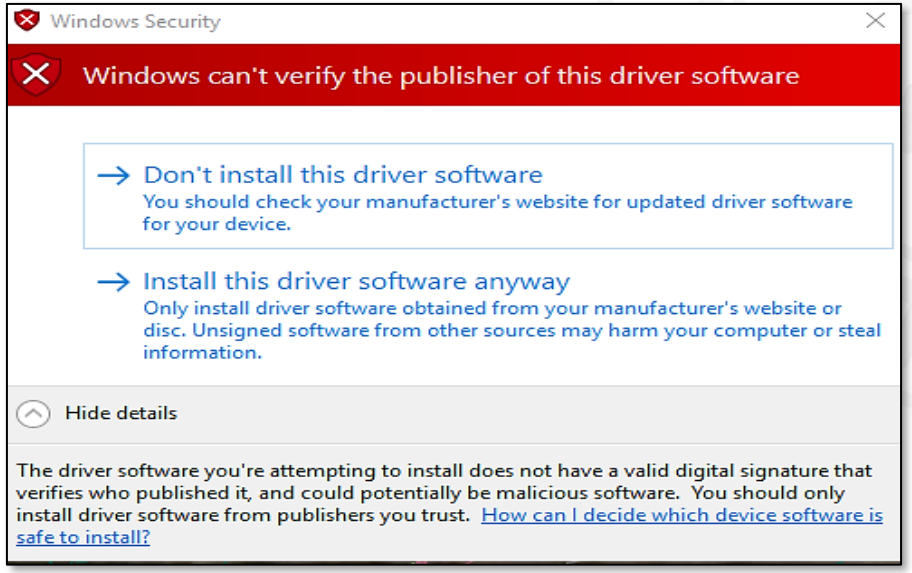
Source: https://www.digicert.com/kb/code-signing/digicert-certificate-utility-to-sign-code.htm

CODE AUTHENTICATION

José Manuel
Redondo López

- **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





< Go to Index

Source: zona-militar.com

STEGANOGRAPHY

Hiding data at plain sight

STEGANOGRAPHY

José Manuel
Redondo López

Message to hide

<secret>
This image is less innocent than he looks. In fact, it's helping to hide 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

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

How to generate a steganogram

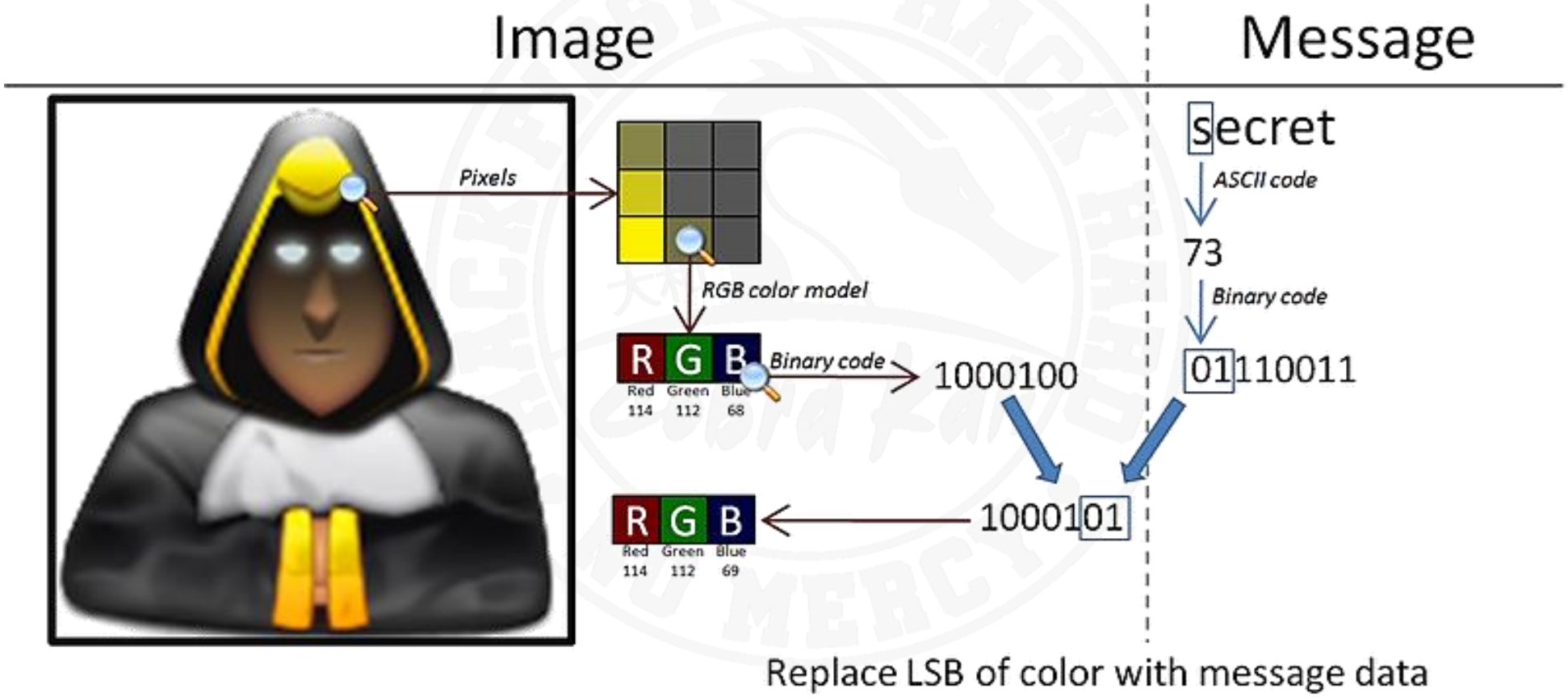
José Manuel
Redondo López

- **Replacing bits on the container object with those corresponding to the information to be hidden**
  - The size of the container file is not altered
  - In case of media files, and thanks to the redundancy and / or excess detail they usually contain, quality is also not reduced in most cases
- **Inserting bits into the container object**
  - Bits of information are added from a certain structural mark of the file (EOF, padding or alignment spaces, metadata, etc.)
  - **It does change the size of the container object**, which may raise suspicions
    - It would be suspicious to have a 16x16 pixel icon that occupies 5 MB, for example
    - The container file must then be appropriately chosen

How to generate a steganogram

José Manuel
Redondo López

- **Image example: LSB (Least Significant Byte) substitution algorithm**



Sources: https://commons.wikimedia.org/wiki/File:Seformatbmp-embedding_full.png

STEGANOGRAM GENERATION

José Manuel
Redondo López

- **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

```
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#
```

```
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

José Manuel
Redondo López

- **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 being transmitted
    - Therefore brute-force attacks (or other variants) can be used against the encryption algorithm
  - If both are used, the probability of transmitting information undetected increases
    - You cannot attack something that you are not aware of! ☺

< Go to Index

Source: zona-militar.com

APPENDIX

Other interesting concepts

PLAUSIBLY DENIABLE ENCRYPTION TECHNIQUES

José Manuel
Redondo López

- **The existence of an encrypted file or message is deniable**
  - An adversary cannot prove that the 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 sensible plaintexts**, depending on the key used
  - This allows the sender to have plausible deniability if forced to give the encryption key
- **This notion 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

Quantum cryptography

José Manuel
Redondo López

- **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 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

José Manuel
Redondo López

- **Cryptography "trending topics"**
  - **Homomorphic**: allows computation over ciphered data
  - **Lightweight**: for devices with low computational power, like IoT
  - **Whitebox**: protection of keys while they reside in RAM
  - **Post-quantum**: 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

To know more…

José Manuel
Redondo López

- **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
  - Seguridad en el protocolo SSL-TLS (free)
    - https://github.com/mindcrypt/libros
  - Esteganografía lingüística y canales encubiertos (free)
    - https://github.com/mindcrypt/libros
- **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

Self-evaluation achievement list: Cryptography

José Manuel
Redondo López

Rank	Concept
	Know the differences between cryptography & steganography
	Know the three main objectives of cryptography
	Understand how ciphering algorithms are classified
	Understand how digital signatures work
	Understand how steganography Works
	Be able to understand and recommend a symmetric key algorithm
	Be able to understand and recommend an asymmetric key algorithm
	Understand the difference between disk encryption and filesystem encryption
	Understand typical usages of hash algorithms and what is a KDF
	Understand how trusted ring (GnuPG) and CAs work and are different from each other
	Understand the different uses of digital certificates and authenticated encryption
	**The guardian of the crypt**: Be able to understand and deploy encryption in typical scenarios

TOPIC 2. CRYPTOGRAPHY APPLICATIONS

Source: zona-militar.com