# PV181 Laboratory of security and applied cryptography

## Symmetric cryptography

Marek Sýs, Zdeněk Říha

Centre for Research on Cryptography and Security

# Before we start

- Log into your account within IS
- Find and download provided materials
- Look where is the openssl  folder
  - Start -> openssl -> version -a

# Goals of Cryptography

- Confidentiality (privacy) - preventing open access
  - **ciphers**
- Authentication:
  1. Entity – identity verification – various (password, MAC, …)
  2. Data origin – identity of message originator – **MAC**
- Integrity - preventing unauthorized modification
  - **hash functions**
- Non-repundation - preventing denial of actions
  - **digital signature**

# Crypto primitives

- **Ciphers** – encryption/decryption of data using **key**
  - Symmetric ciphers – **same** key for enc/dec
  - Asymmetric ciphers – **different** key for enc/dec
- **Random number generators** (RNGs)
  - Key generation
- **Hash functions** – “unique” fingerprint of data

- Based on previous: MAC, PBKDF, Digital signature

# Standards

Primitives are defined in various types of standards:

- FIPS PUB 197 – AES block cipher
- RFC1321 – md5 hash function
- NIST SP,…

**Test vectors**: defined output to test implementation

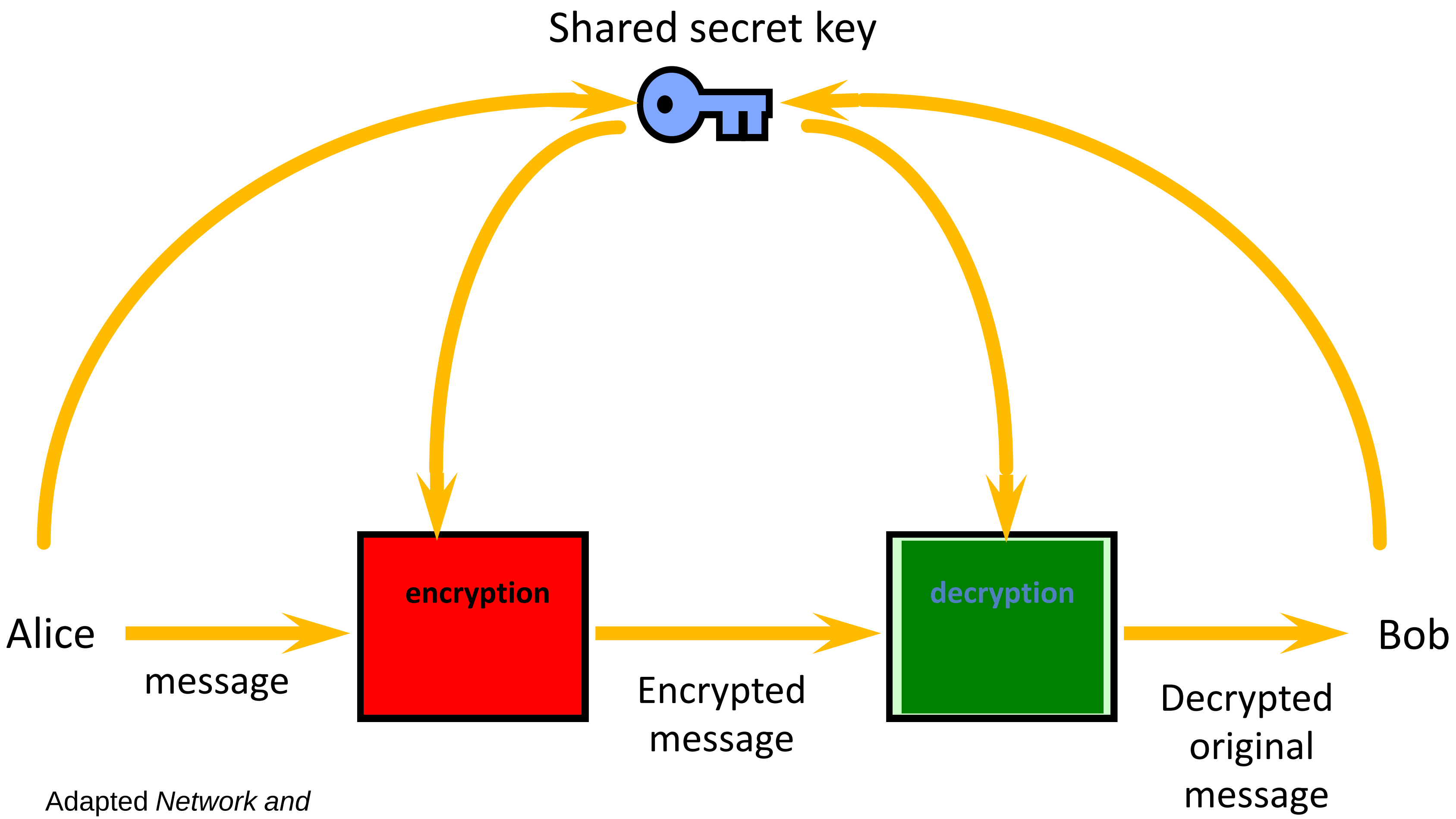
- MD5 ("") = **d41d8cd98f00b204e9800998ecf8427e**

# Ciphers: Kerckhoffs' principle

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

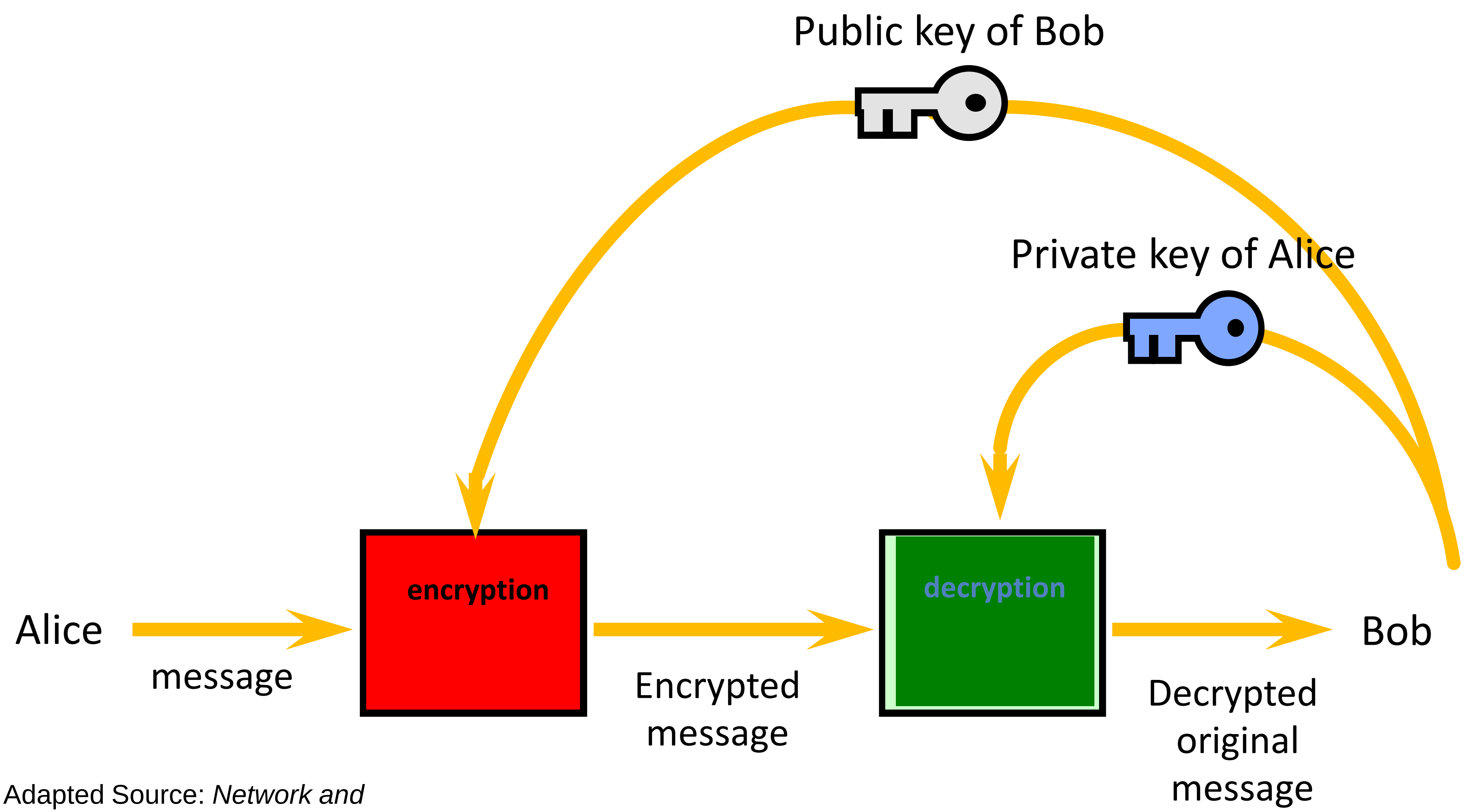
- I.e. only the **key** should be kept secret, not the algorithm.

# Symmetric cryptosystem




Adapted *Network and Internetwork Security* (Stallings)

# Asymmetric cryptosystem

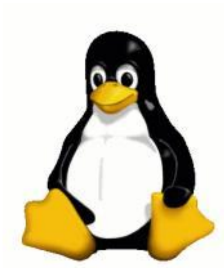


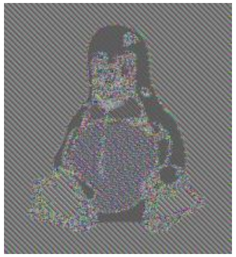

Adapted Source: *Network and Internetwork Security* (Stallings)

# Block cipher

- Input divided into blocks of fixed size (e.g 256 bits)
  - Padding - message is padded to complete last block

- Different modes of operation:
  - Insecure basic ECB mode – leaks info
  - Secure modes: CBC, OFB,CFB,CTR,..





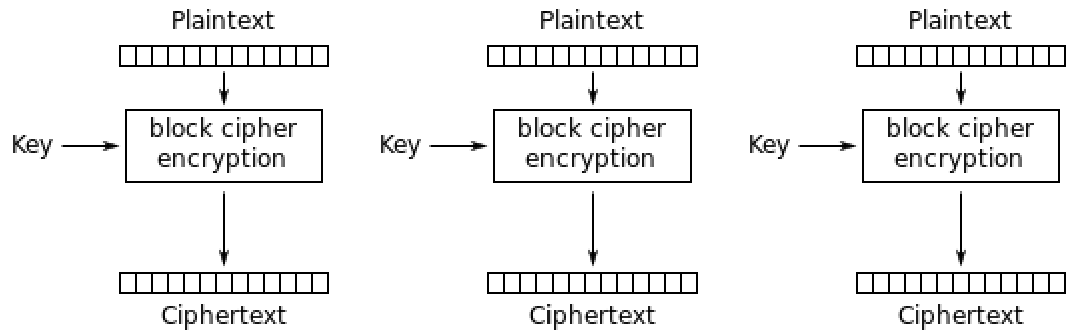
- CBC, OFB,CFB need initialization
  - Initialization vector (IV) – must be known

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

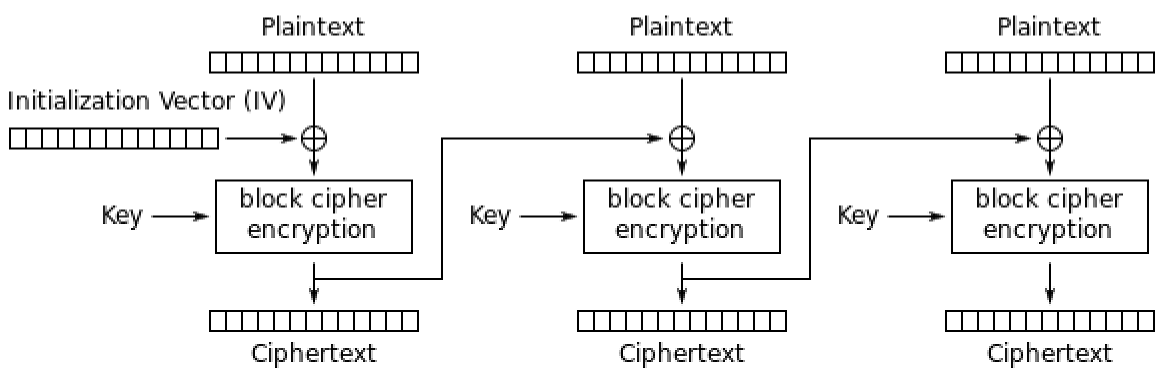
# Block ciphers - padding

| *Standard* | *method* |
| --- | --- |
| ANSI X.923 | `... \| DD DD DD DD DD DD DD DD \| DD DD DD DD` **`00 00 00 04`** `\|` |
| ISO 10126 | `... \| DD DD DD DD DD DD DD DD \| DD DD DD DD` **`81 A6 23 04`** `\|` |
| PKCS7 | `... \| DD DD DD DD DD DD DD DD \| DD DD DD DD` **`04 04 04 04`** `\|` |
| ISO/IEC 7816-4 | `... \| DD DD DD DD DD DD DD DD \| DD DD DD DD` **`80 00 00 00`** `\|` |
| Zero padding | `... \| DD DD DD DD DD DD DD DD \| DD DD DD DD` **`00 00 00 00`** `\|` |

# Block ciphers: ECB vs CBC mode




Electronic Codebook (ECB) mode encryption




Cipher Block Chaining (CBC) mode encryption

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

# Random number generators

- Used to generate: keys, IV, …

1. Truly RNG - physical process
   - aperiodic, slow
2. Pseudo RNG (PRNG) – software function
   - deterministic, periodic, fast
   - initialized by **seed** – fully determines random data

- Combination often used:
  - truly RNG used to generate **seed** for PRNG
  - dev/urandom, dev/random in Linux, **Fortuna** scheme

# Hash function

- **Cryptographic** hash function
- Input of arbitrary size
- Output of fixed size: n bits (e.g. 256 bits).
- Function is not injective (there are “collisions”).
- Hash is a compact representative of input (also called imprint, (digital) fingerprint or message digest).
- Hash functions often used to protect integrity. First the has is computed and then only the hash is protected (e.g. digitally signed).

# Hash function properties

- One-way property
  - It is easy to calculate **h(x)** for arbitraty **x.**
  - In a reasonable time it is not possible for the fixed **y** to find **x**, such that **h(x) = y**.

- Collision resistance
  - (**weak**): In a reasonable time it is not possible for a given **x** to find **x'** (**x ≠ x'**) such that **h(x) = h(x')**.
  - (**strong**): In a reasonable time it is not possible to find any **x, x'** such that **h(x) = h(x')**.

# Cryptographic hash functions

- MD5: output 128 bits
  - Still used although not considered secure at all
  - Broken: efficient algorithm for finding collisions available
  - 128-bit output not considered secure enough
- RIPEMD
  - Output : 128, 160, 256 or 320 bits
  - Less frequently used
- Whirlpool
  - Output: 512 bits
  - Based on AES
  - Recommended by NESSIE project
  - Standardized by ISO

# Secure Hash Algorithm (SHA)

- **SHA-1**
  - NIST standard, collision found in 2016, 160 bits hash
- **SHA-2**
  - function family: SHA-256, SHA-384, SHA-512, SHA-224
  - defined in FIPS 180-2
  - Recommended
- **SHA-3**
  - New standard 2015
  - Keccak sponge function family: SHAKE-128, SHA3-224, …
  - defined in FIPS 202, used in FIPS-202, SP 800-185
  - Recomended

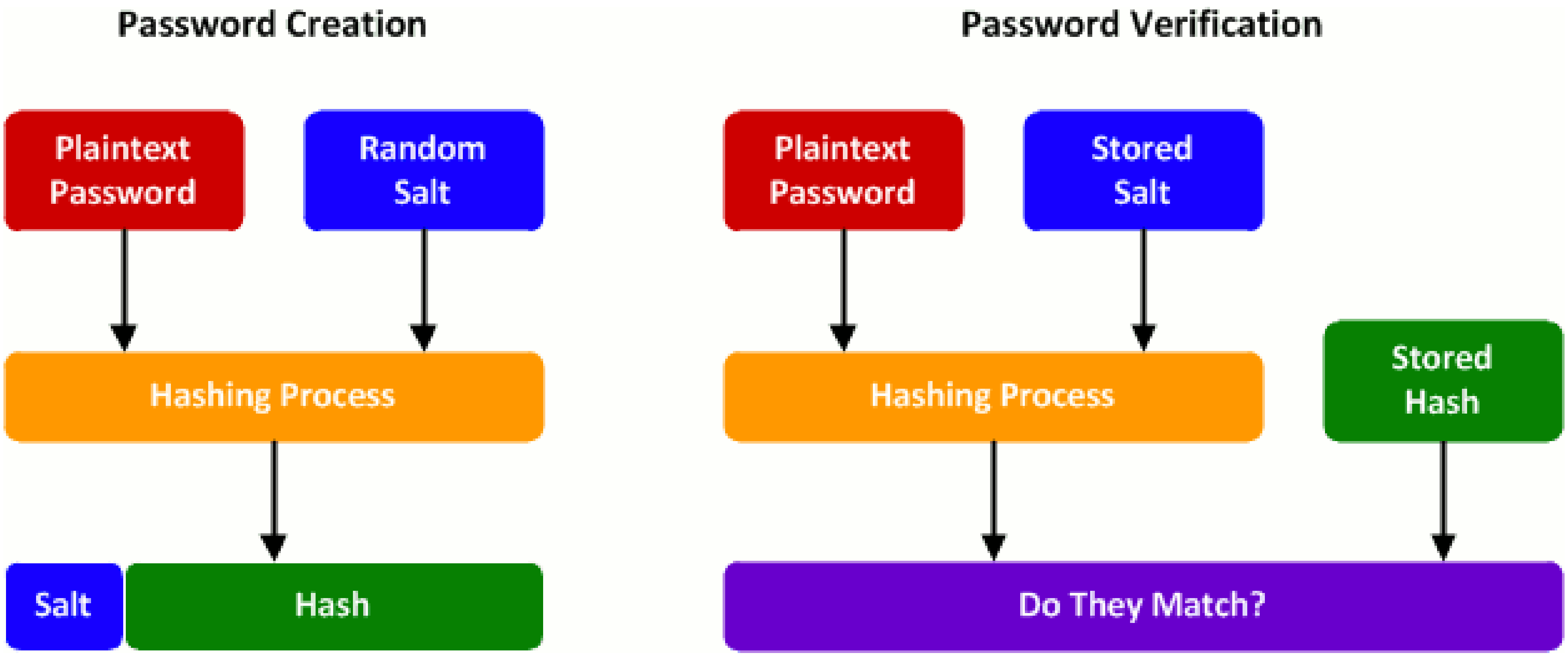
# Hash functions - examples

- MD5
  - Input: „Autentizace“.
  - Output: 2445b187f422458303788851 1d5411c7 .
  - Output 128 bits, written in hexadecimal notation.
  - Input: „Cutentizace“.
  - Output: cd99abbba3306584e90270bf015b36a7.
  - A single bit changed in input → big change in output, so called “Avalanche effect”
- SHA-1
  - Input: „Autentizace“.
  - Output: 647315cd2a6c953cf5c29d36e0ad14e395ed1776
- SHA-256
  - Input: „Autentizace“.
  - Output: a2eb4bc98a5f71a4db02ed4aed7f12c4ead1e7c98323fda8ecbb69282e4df584

# Password protection
# password hashing & salting

1. Clear password could be stolen:
   - store hash of password
     **hash** = H(password)
   - Checking: password is correct if **hash** matches
2. Attack (brute force or dictionary)
   - trying possible passwords “aaa”, “aab”…“zzz” – N tests
   - N test for single but also for 2,3,… passwords **!!!**
3. Slow down attack - increase password size:
   - random “**salt**” is added to password,

# Password protection
# password hashing & salting

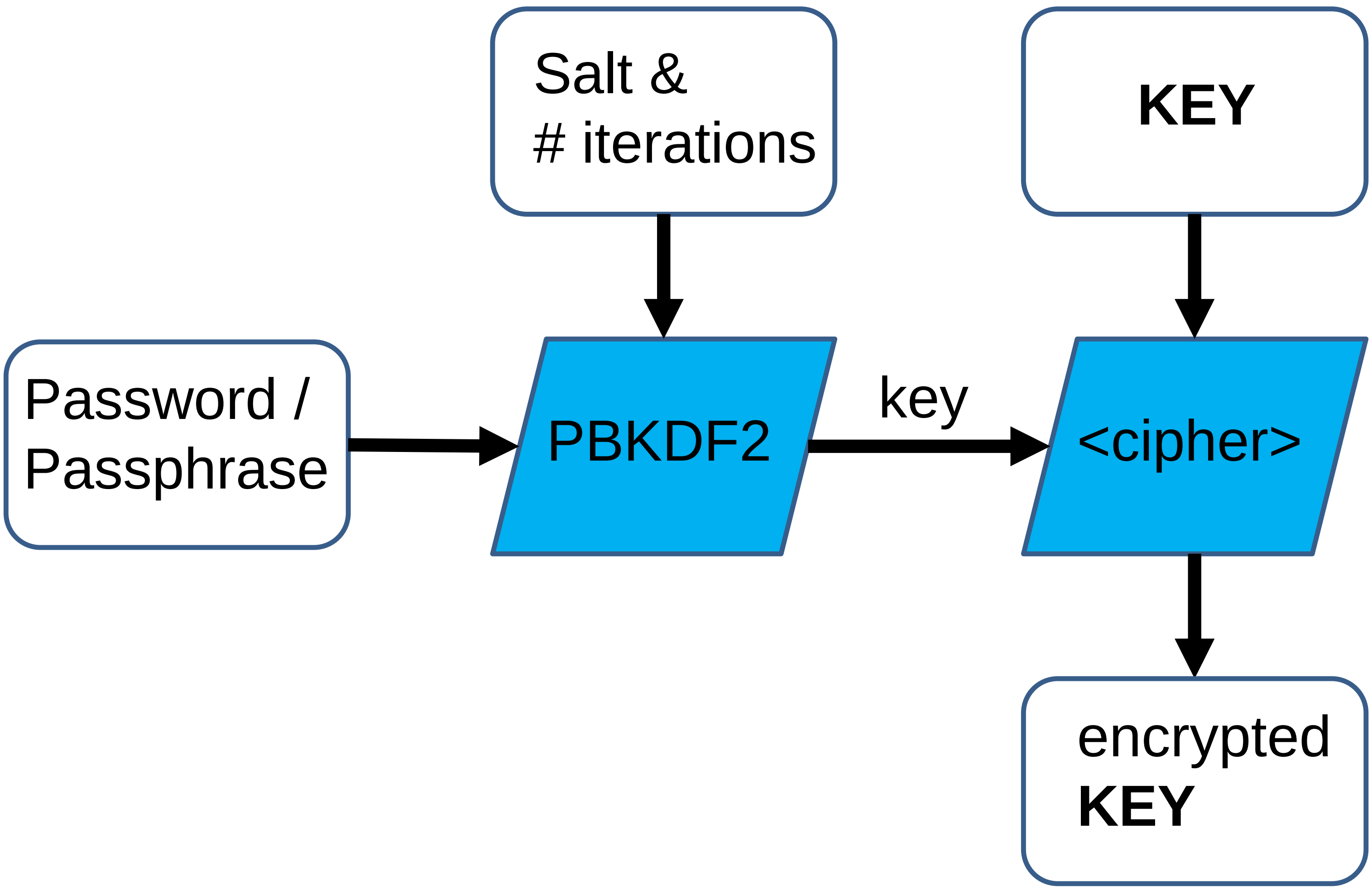



Source: http://blog.conviso.com.br/worst-and-best-practices-for-secure-password-storage/

# Key protection

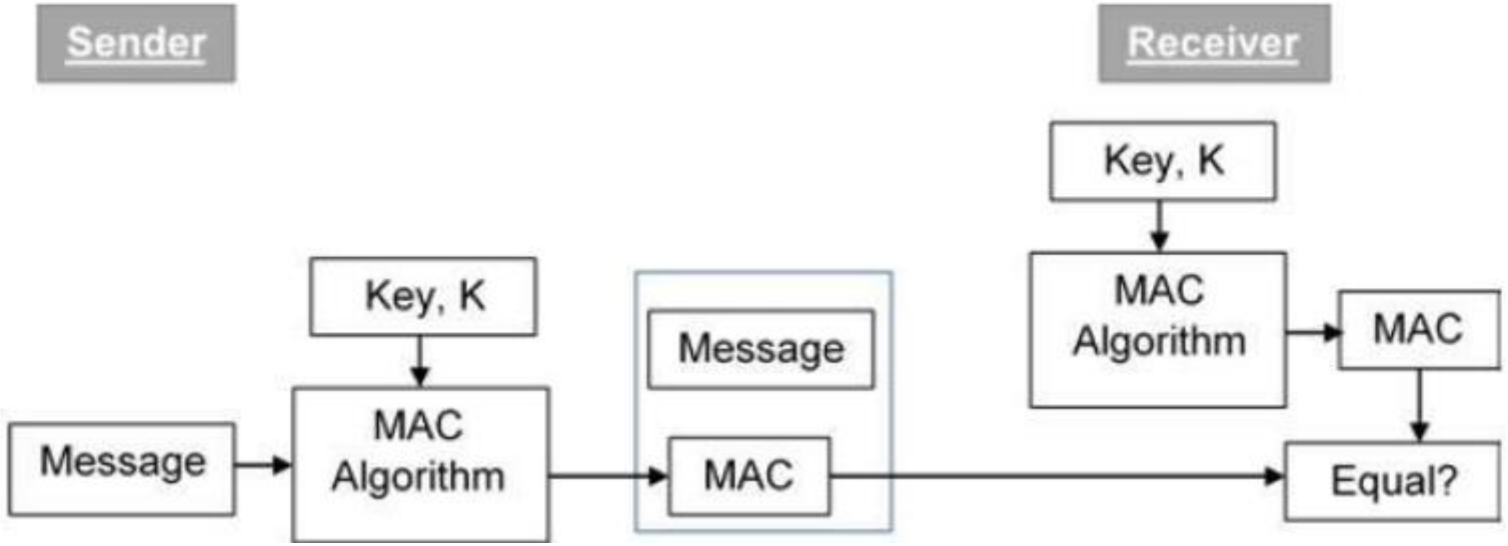
- Encrypt **key** K (using cipher and other **k**):
  - Key **k** typically derived from password
- Insufficient entropy of passwords
  - E.g. only 17000 guesses for ***
  - **salt** protects many passwords not single (is stored)
- Password Based Key Derivation Function (PBKDF):
  - 2 types PBKDF2 is newer (PKCS#5 )
  - slow down hash function
  - iterate hash function ***c*** times - $\text{Key} = H^{\boldsymbol{c}}(pwd \mid \boldsymbol{salt})$:

# PBKDF2


Salt &
# iterations
KEY
Password /
Passphrase
PBKDF2
key
<cipher>
encrypted
KEY

# Message authentication code (MAC)

- – Based on block cipher (MAC) or hash function (HMAC)
  – Key + message → algorithm → fixed size block MAC




Source: https://www.tutorialspoint.com/cryptography/message_authentication.htm

# RSA: mathematics

- Prime multiplication is simple & Factorization of integers is computationally intensive.
- We choose randomly 2 primes and compute n and φ(n) :
  - **p, q**
  - **n** = **p·q**
  - **φ(n) = (p-1)(q-1).**
- **e** is chosen such that **gcd(e, φ(n)) = 1.**
- We compute **ed = 1 (mod φ(n)).**

- Public key: **n, e.**
  Private parameters: **p, q, d.**
  Private key: **d.**

- For RSA with 1024 bit **n**, the encrypted message will be 1024 bit long.

# RSA: example

- Intentionally small numbers (such cryptosystem is **not** secure).

- We generate parameters:
  - $p = 17$, $q = 7$,
  - $n = pq = 119$,
  - $\varphi(n) = 16 \times 6 = 96$.
- Public exponent is selected **e = 3,5**, equation is solved (**3 can not be**) **ed = 5d = 1 (mod 96)** to have **d = 77**.

- The public key: **(n = 119, e = 5)**,
  The private key: **d = 77**.
- Encryption/decryption:
  - Message **m** = 'C' = 65
  - Encryption **m'** = $65^5$ mod 119 = 46.
  - Decryption m = $46^{77}$ mod 119 = 65

# Links

- SHA1 collision:
  - https://shattered.io
- Salting password:
  - https://crackstation.net/hashing-security.htm
- OpenSSL
  - Manual: https://www.openssl.org/docs/man1.0.2/
  - https://wiki.openssl.org/index.php/Command_Line_Utilities
  - https://www.madboa.com/geek/openssl/