PV181 Laboratory of security and applied cryptography

Symmetric cryptography

Marek Sýs, Zdeněk Říha

CROCS

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

OpenSSL basics

Manual pages: www.openssl.org/docs/manmaster/

Basic usage: **OpenSSL commands params**

Help: non-recognized command (e.g. –help)

Basic commands:

1. **dgst** – hashing, MAC
2. **enc** – encryption

How to check implementation of crypto primitive?

- **Test vectors**
  - standards define outputs for selected inputs
- MD5 defined in RFC1321:
  - MD5 (**""**) = **d41d8cd98f00b204e9800998ecf8427e**
  - MD5 ("message digest") = **f96b697d7cb7938d525…**
- AES defined in FIPS 197
  - Plaintext: **00112233445566778899aabbccddeeff**
  - Key: **000102030405060708090a0b0c0d0e0f**
  - Ciphertext **69c4e0d86a7b0430d8cdb78070b4c55a**

How to check hash of data?

- **Hashed**, **encrypted, …** data are typically binary
  - – not human readable
- Human readable formats
  - Hex – byte encoded by 2 chars from charset **0-9**, **A-F**
  - Base64 – each **6** bits encoded by 1 char
    - – charset is **A-Z**, **a-z**, **0-9, +, /**
    - – as padding **=** is used
      - 0,1 or 2 times (no padding, “=“ or “==“)

Hash function - properties

- **Cryptographic** hash function
- Input of arbitrary size
- Output of fixed size: n bits (e.g. 256 bits).
- Function is not injective (there are “collisions”)
  - should be hard to find them
- 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 functions - examples

- MD5
  - Input: „Autentizace“.
  - Output: 2445b187f4224583037888511d5411c7 .
  - 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

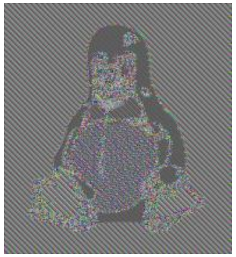
Secure Hash Algorithm (SHA)

- **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
  - Recommended

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

Random number generators

- Used to generate: keys, IV, …

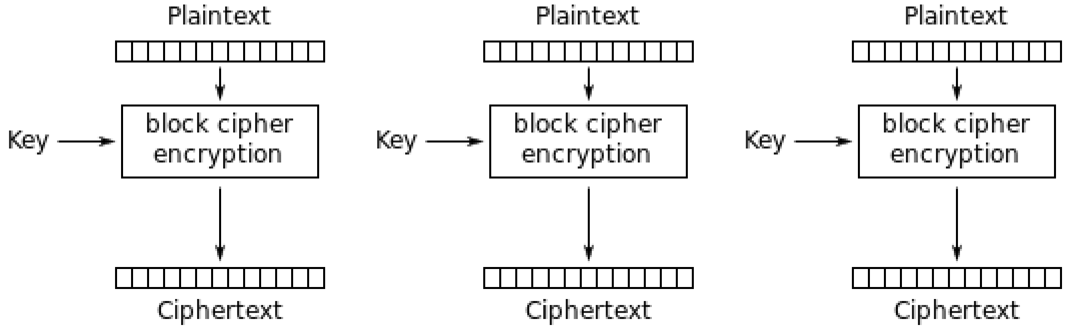
1. Truly RNG - physical process
   - aperiodic, slow
2. Pseudo RNG (PRNG) – software function
   - deterministic, periodic, fast
   - initialized by **seed** – fully determines random data

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

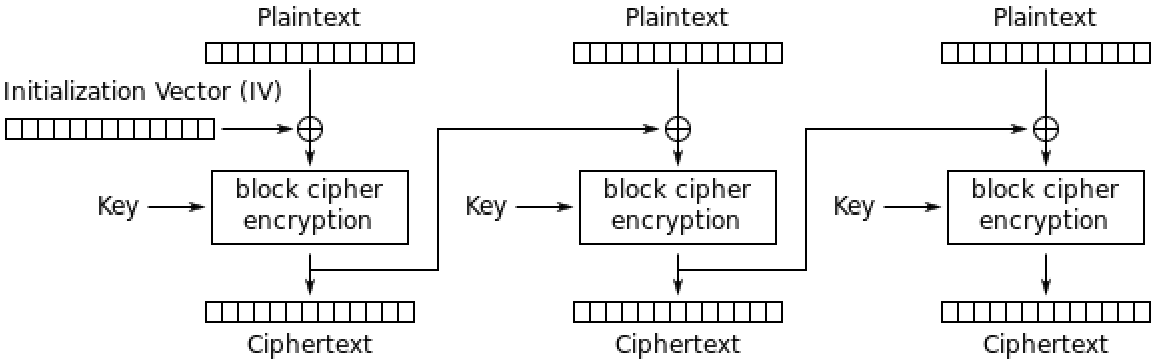
Block ciphers - padding

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

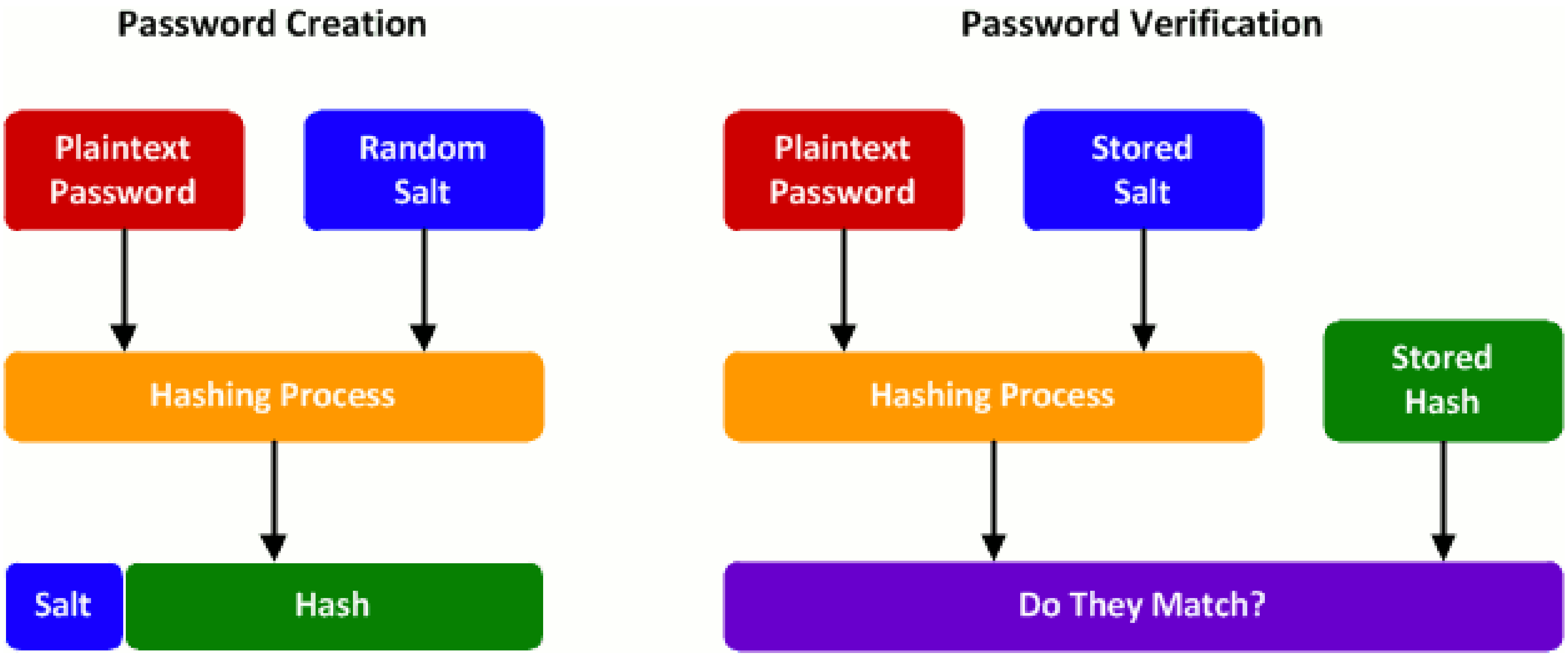
Password protection
password hashing & salting

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. Salt - random string (salt) added to password

   hash = H(salt | password)
   - protects many passwords not one (salt also stored)

Password protection
password hashing & salting



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

Key protection

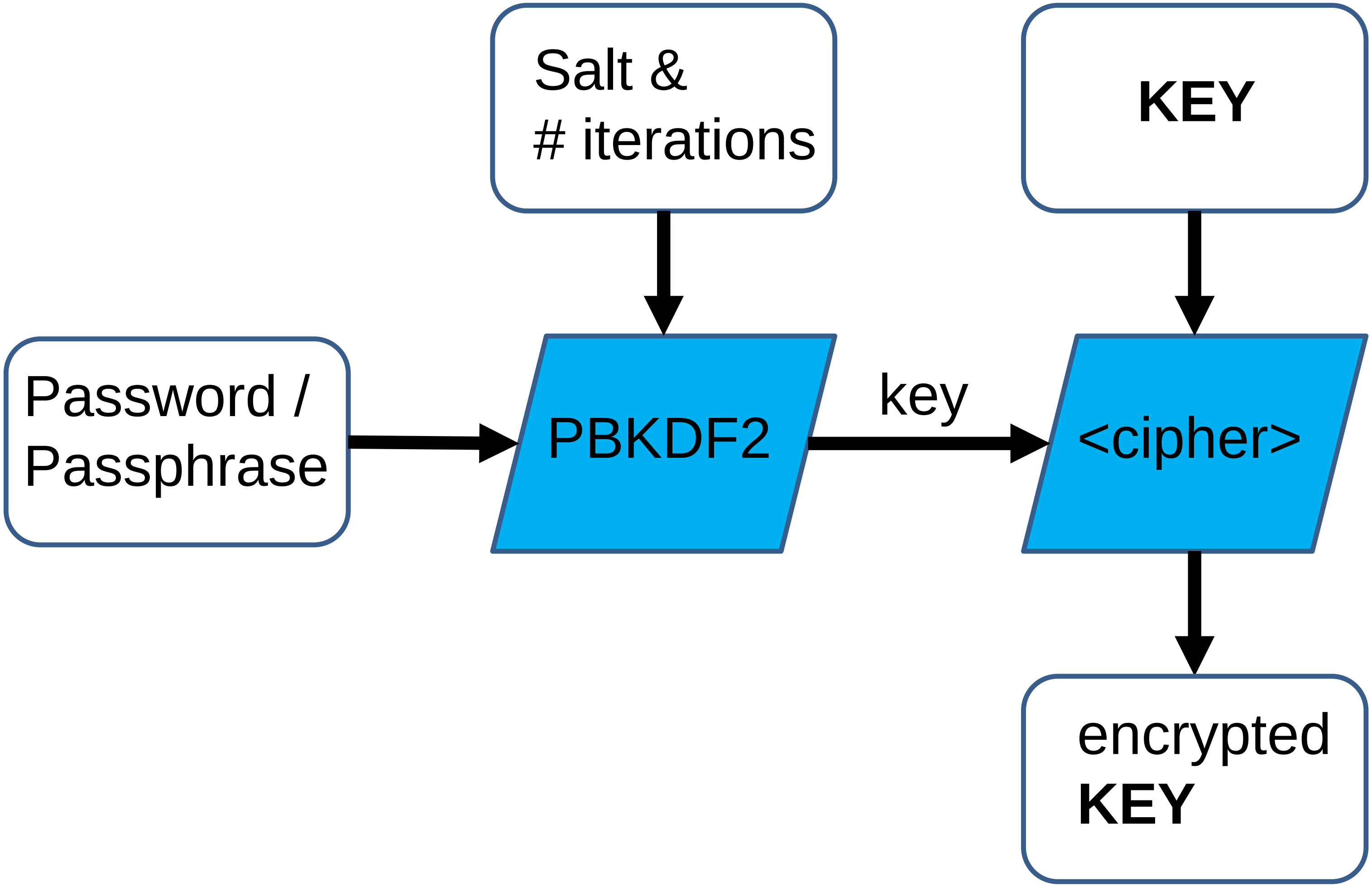
- Encrypt **key** (using cipher and other key **k**)
  - – Key **k** typically derived from password

4. Password based key derivation function (PBKDF):
   - – 2 types - PBKDF and newer PBKDF2 (PKCS#5)
   - – slow down hashing of passwords – hash of hash of hash…

$$\mathbf{k} = \boldsymbol{H}^{c}(\boldsymbol{salt} \mid \boldsymbol{pwd})$$

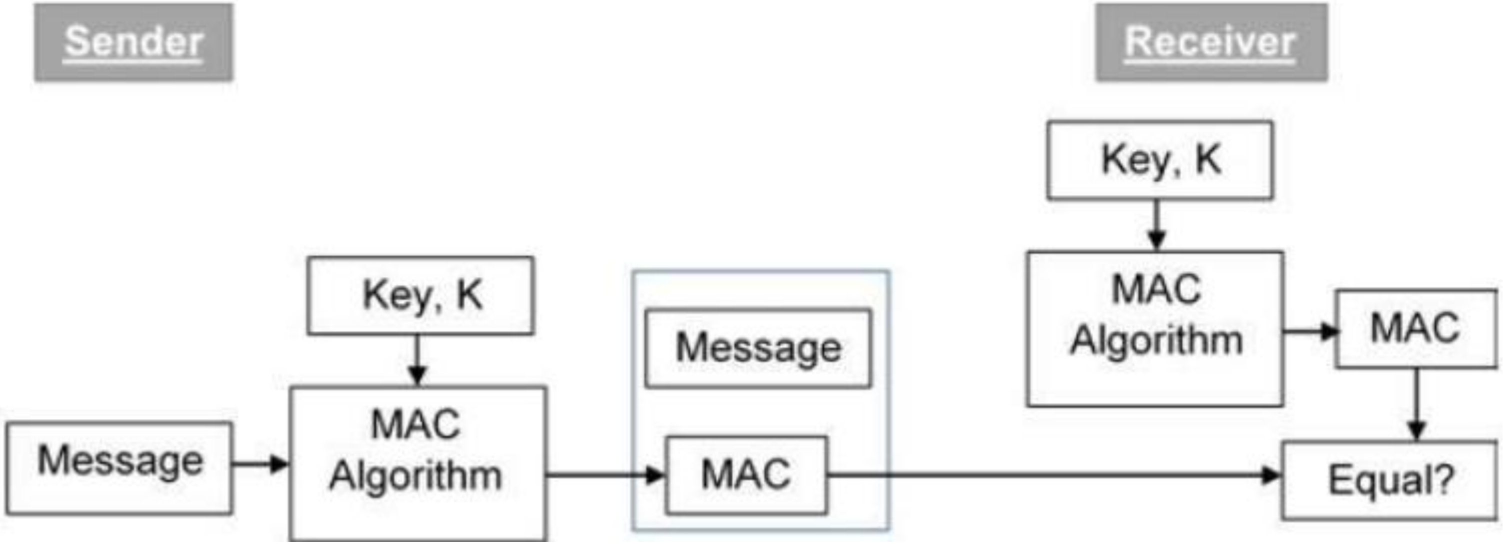
   - – Attacker is c times slower / need c times more resources

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

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/