# (user) authentication:

## On the extreme weakness of passwords

# Password or secret key?

➔**Authentication**

⇨Prove I know a password or a secret

➔**Passwd or secret? conceptually the same**

⇨A password is (should be) a secret

➔**Practically there is a HUGE difference**

⇨Secret ley: a random string of N bit

➔Probability to guess it = 1 out of $2^N$

⇨Password: a **low-entropy** string!!

➔Probability to guess it:
MUUUUCH lower than n = 1 out of $2^N$

(difficilmente una password è randomica!)

# Password: four major problems

➔**Password overload**

⇨users VERY often reuse passwords across different sites

➔**Restricted charset**

⇨Not all possible characters are used!

➔**Low entropy**

⇨Password meant to be remembered: not nearly random!

➔**Predictability (and dictionary attacks)**

⇨Bad habits in choosing passwords

⇨Use human-friendly rules to “generate” passwords

# Issue 1: password overload

- **Average # of digital accounts per person in US: 130 (!!)** (1 persona ≈ 130 Account)
  - 2017 statistics
- **38% of users reuse identical password across sites**
  - Source: 2018 scientific paper (*)
- **21% of users modify password…**
  - But «rules» used are VERY predictable:
    - 30% cracked in 10 or less attempts
    - 46.5% cracked in 100 or less attempts
  - Modification examples from (*):

| | |
|---|---|
| danilo | → da11nilo |
| butcher69 | → butcherboy |
| Martin1 | → martin1 |
| fishkakemmm | → fishkake333 |

**VERY SERIOUS SECURITY ISSUE: Cross-site breach!**

**Password reuse: ever many years after a breach**



(*) https://people.cs.vt.edu/gangwang/pass

# Issue 2: restricted charset

- **Secret key = 8 bytes**
  - 1 byte = 256 possible values
    - → Probability of guess = 1/256
  - Probability of guessing all 8 bytes
    - → $1/256 * 1/256 * \ldots$ (8 times) $= 1 / 256^8$
      $= 1 / 18.446.744.073.709.551.616$
  - Assuming 66M guesses/s
    - » We will see later on why this number!
    - → Average guess time = $1.8 \times 10^{19} / 6.6 \times 10^7 / 2$ seconds = **4431 years**
- **Passwd = 8 bytes**
  - But if you use only low case letters and numbers?
  - 1 byte = 36 possible values!
  - Probability of guessing all 8 bytes password
    - → $1 / 36^8 = 1 / 2.821.109.907.456$
    - → Average guess time = $2.8 \times 10^{12} / 6.6 \times 10^7 / 2$ seconds = **5.9 hours**!!

(quanto è random la pw)

# Issue 3: low entropy

## ➔With (very!) rare exceptions, passwords (human generated strings) do not «look» random

⇨10 letter truly random examples computed using https://www.random.org/strings/

➔wykvpdslzb, iuxdacetyu, sermfokoia, zpkwtrehjv, grsshdnixc, axqeudtijf, jzmkmhlsop , le pw non sono realmente così! $26^{10}$ combinazioni

## ➔How to «quantify» randomness?

Shannon entropy

esempio esame: Pin Code Entropy vs |DD|MM| format

PIN code: 4 elementi, 10 char: $10^4$ uniforme → $H(X) = \left(10^4 \cdot \frac{1}{10^4} \cdot \log_2 \frac{1}{10^4}\right) \cdot (-1) = 13{,}28$ bit

(casi totali; mi dice n. bit che servono) $= \log_2(10^4)$

date: equiprobabili ~> $\log_2[366] = 8$ bit, ho perso 5 bit,

ogni 3 bit ho $2^3 = 8$, $2^2 = 4$ → $4 \cdot 8 = 32$,

ci metto 32 volte meno ÷

D D M M

Renderei tempo a vedere che valori posso ottenere DD e MM (es: $M_1 = 0,2$ $M_2 = 2,9$ → non 29!!)

posso avere 366 date diverse

• 8 bytes con { C maius (26), c minusc (26), N numeri (10) } differenza tra truly random vs CccccNN, entropy?

• truly: 62 valori: 8 bit → $62^8$ $\xrightarrow{H}$ $\log_2(62^8)$ [equiprobabili] $= 62^8 \cdot (-1) \cdot \frac{1}{62^8} \log_2\left(\frac{1}{62^8}\right)$ ≅ 47 bit of security

• C · ccccc · NN → $H(x) = \log_2(26^5 \cdot 10^2) = 34$ bit, ho 13 bit di differenza, so che 10 bit $26 \cdot (26)^5 \cdot 10^2$ possibili è un fattore di 1000, 3 bit → $2^3 = 8$ → 8000, quindi un attacco brutto richiede 8000 volte meno del truly

NB 8.3 bit ≈ 10

tutte pw di len = 10

es 11

# Entropy

$n = 26^{10}$ pw diverse.

$p_i = \frac{1}{26^{10}}$ (truly random)

## ➔Let X be a discrete random variable

⇨Discrete random variable: a quantity whose outcome is $x_1, x_2, \ldots x_n$ with probability $p_1, p_2, \ldots p_n$

entropy = 0 = no random, prevedibile

## ➔Entropy: $H(X) = -\sum_i p_i \log_2 p_i$

⇨Measured in bit = quantity of information carried by X

⇨Example: coin flip: $x_1$ = face A, $x_2$ = face B, $p_1 = p_2 = 1/2$

➔$H(X) = -2 \cdot \left(\frac{1}{2}\log_2 \frac{1}{2}\right) = -\log_2 \frac{1}{2} = \log_2 2 = 1$ bit information

⇨Example: unbiased dice: 6 equiprobabile outcomes

(2 alterm) 2 bit < H(X) < 3bit (8 alternative)

➔$H(X) = -6 \cdot \left(\frac{1}{6}\log_2 \frac{1}{6}\right) = \log_2 6 = \mathbf{2.58}$ bit

truccata { testa $P_t = 1$ / croce $P_c = 0$ → $H(X) = -1 \log_2 1 - 0 = 0$ bit (deterministic)

# Why this mathematical definition?
## (Shannon Information theory, 1948)

- **"informational value" of $x_i$ depends on how much $x_i$ is "unexpected"**
  - ⇨ The lower is its probability $p_i$, the more it is "surprising"
  - ⇨ information content = function of $1/p_i$

compressione legata ad entropia, più random è, più difficile da compattare.

- **Convenient "translation" in bits via log2 (-log2, to keep it positive)**
  - ⇨ event happening with prob. $1/2^b$ happens on average once every $2^b$
    - → information content = b bits: $-\log_2 \frac{1}{2^b} = \log_2 2^b = b$

- **Entropy: average value of the information content**
  - ⇨ $H(X) = E[IC(X)] = \sum_i p_i \, IC_i = -\sum_i p_i \log_2 p_i$

media  p  information content

# Entropy and predictability

➔ **Entropy: a quantitative measure of «how predictable» is a random event!**

- ⇨ Assume $N=2^b$ possible outcomes → $b = \log_2 N$
  - →Entropy = 0 → event is deterministic: I can tell which event will happen
  - →Entropy = b → no way to predict: all events are equiprobabile → maximum extropy
  - →Entropy between 0 and b: it is more likely that some events will occur than others

➔ **Example: biased bit: $p_0=1/4$, $P_1=3/4$**

- ⇨ Outcome 1 is more likely than 0 → predictability
- ⇨ $H(X) = -\frac{1}{4}\log_2\frac{1}{4} - \frac{3}{4}\log_2\frac{3}{4} = 0.81 < 1$
- ⇨ Consequence: every transmitted bit carries 0.81 bits of information
- ⇨ Consequence: source emitting stream of such bits (independent) CANNOT be compressed of more than 19%

# Test your understanding
## (Entropy and dependence)

- **Bit flip: $X_k$ = {0,1} with probability ½**
  - Entropy = 1 bit

(indipendenti)

- **Entropy of sequence of independent r.v.: $X_1$ $X_2$ $X_3$**
  - Entropy = 3 bits

- **What if they are dependent?**
  - $X_1$ is bit flip, but $X_2$ and $X_3$ have same value as $X_1$
    - » Hence they are still {0,1} with probability ½ but depend on the previous value
  - Entropy = ???
  - **Answer: 1 bit!**
  - Why? Two trivial explanations
    - Information is only carried by one bit (bit $X_1$)
    - r.v. Y = $X_1$ $X_2$ $X_3$ has only two outcomes; 000 and 111, with p=½

# Prediction and Entropy of Printed English

By C. E. SHANNON

(*Manuscript Received Sept. 15, 1950*)

A new method of estimating the entropy and redundancy of a language is described. This method exploits the knowledge of the language statistics possessed by those who speak the language, and depends on experimental results in prediction of the next letter when the preceding text is known. Results of experiments in prediction are given, and some properties of an ideal predictor are developed.

decrease → entropy

anche se ho 26 lettere, più ho lettere e più predico le successive: A → 1/16 → ~> 1/8 se entropia è 3

| Column | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 100 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Upper. . . . . | 4.03 | 3.42 | 3.0 | 2.6 | 2.7 | 2.2 | 2.8 | 1.8 | 1.9 | 2.1 | 2.2 | 2.3 | 2.1 | 1.7 | 2.1 | 13 |
| Lower. . . . . | 3.19 | 2.50 | 2.1 | 1.7 | 1.7 | 1.3 | 1.8 | 1.0 | 1.0 | 1.0 | 1.3 | 1.3 | 1.2 | .91 | 1.2 | .6 |

➔ **Couple of bits/letter!! MUCH lower than uniform random letters – brute-force crack:**

"Se pw umana ha 10 lettere mi aspetto entropia 2 bit. lettera = 20

- ⇨ random 10 letters → $2^{(4.7 \times 10)} = 2^{47}$
- ⇨ Human-generated → about $2^{(2 \times 10)} = 2^{20}$
  factor of $2^{27}$ ~ 100M times less than computer-generated!
  But is worse than this, see next issue 4

# Issue 4: dictionary attacks

➔**Password guessing heuristics; improve brute-force**

- ⇨Use set of **«common» words** (dictionary)
- ⇨Use **«public» password** databases
  - →Obtained from breaches (e.g. published by hacking groups)
- ⇨Use dictionaries **customized to targeted victim**
  - →Her interests, her context, etc
  - →Usually complemented by social engineering
- ⇨Use password modification rules (more later)

➔**Types of attack**

- ⇨**Online** → easy to defend (limit #attempts → account lockout)
- ⇨**Offline** (against hashed pw) → no defence
  - →(except chosing strong passwords)

# Some statistics

Se ho 1M di vittime, meglio provare prima psw ∀ utente, poi 2ª... invece che provare tutte le psw per utente e poi passare al 2.

## ➔Common passwords choice

- ⇨25% similar to TOP-20 (≈ provate psw a cascata in vers)
  - ➔**That's why password spraying attacks are VERY effective…**
- ⇨16% first name
- ⇨4% "*password*" variant

## ➔Common passwords length

- ⇨26% length of 6 byte
- ⇨19% length of 7 byte
- ⇨20% length of 8 byte

| | |
|---|---|
| 1 | 123456 |
| 2 | 12345 |
| 3 | 123456789 |
| 4 | Password |
| 5 | iloveyou |
| 6 | princess |
| 7 | rockyou |
| 8 | 1234567 |
| 9 | 12345678 |
| 10 | abc123 |

# More detailed stats

**source: David Malone, Kevin Maher, Investigating the Distribution of Password Choices, 2011**

**➔4 datasets analyzed**

| Site | #users | #pass | $\frac{\#pass}{\#users}$ |
|---|---|---|---|
| hotmail | 7300 | 6670 | 0.91 |
| flirtlife | 98930 | 43936 | 0.44 |
| computerbits | 1795 | 1656 | 0.92 |
| rockyou | 32603043 | 14344386 | 0.44 |

32 Mln users → 14 mln pass

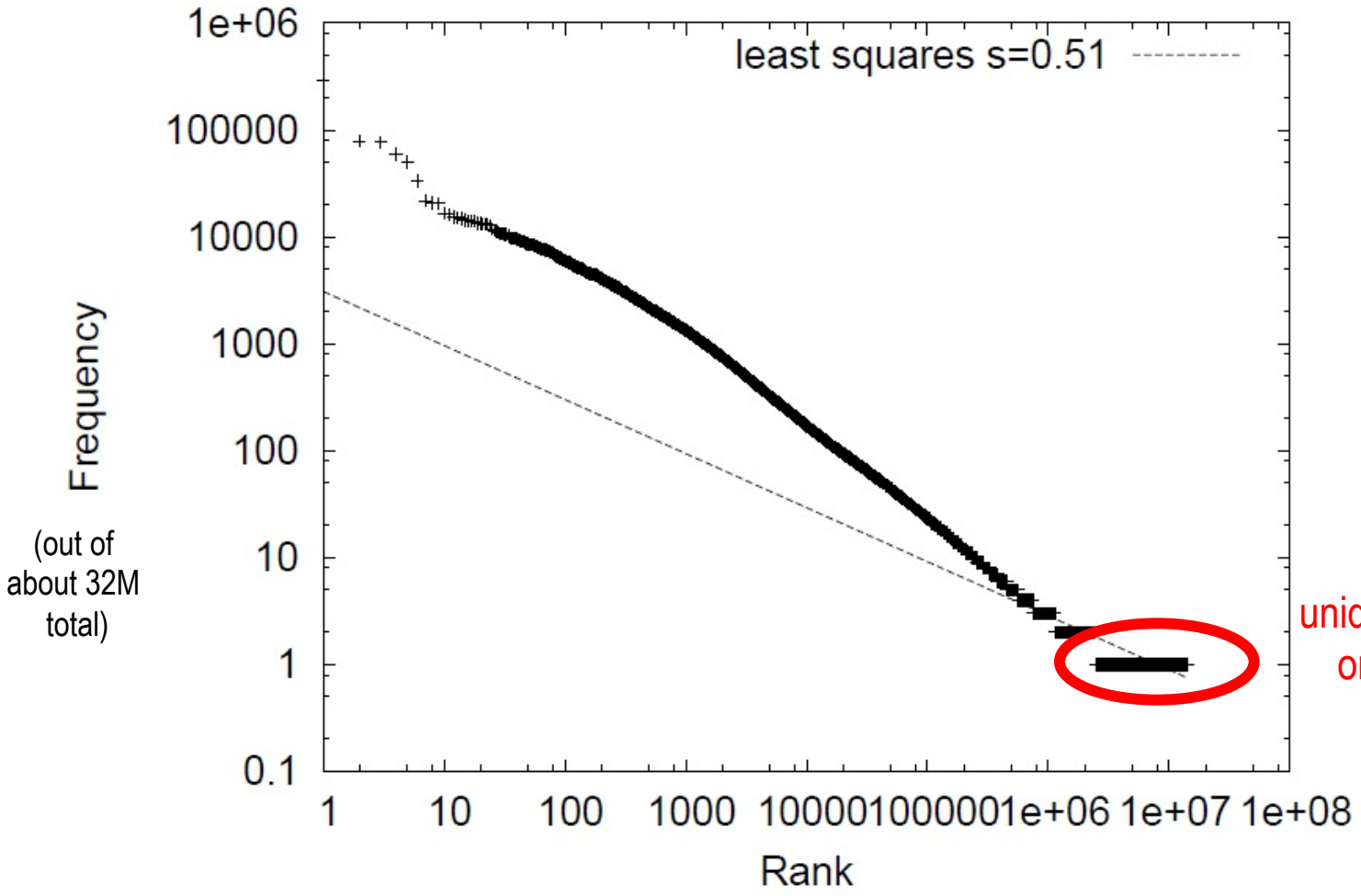
**➔Top 10 per each dataset**

| Rank | hotmail | #users | flirtlife | #users | computerbits | #users | rockyou | #users |
|---|---|---|---|---|---|---|---|---|
| 1 | 123456 | 48 | 123456 | 1432 | password | 20 | 123456 | 290729 |
| 2 | 123456789 | 15 | ficken | 407 | computerbits | 10 | 12345 | 79076 |
| 3 | 111111 | 10 | 12345 | 365 | 123456 | 7 | 123456789 | 76789 |
| 4 | 12345678 | 9 | hallo | 348 | dublin | 6 | password | 59462 |
| 5 | tequiero | 8 | 123456789 | 258 | letmein | 5 | iloveyou | 49952 |
| 6 | 000000 | 7 | schatz | 230 | qwerty | 4 | princess | 33291 |
| 7 | alejandro | 7 | 12345678 | 223 | ireland | 4 | 1234567 | 21725 |
| 8 | sebastian | 6 | daniel | 185 | 1234567 | 3 | rockyou | 20901 |
| 9 | estrella | 6 | 1234 | 175 | liverpool | 3 | 12345678 | 20553 |
| 10 | 1234567 | 6 | askim | 171 | munster | 3 | abc123 | 16648 |

# More detailed stats

source: David Malone, Kevin Maher, Investigating the Distribution of Password Choices, 2011

## ➔Rank/frequency plot

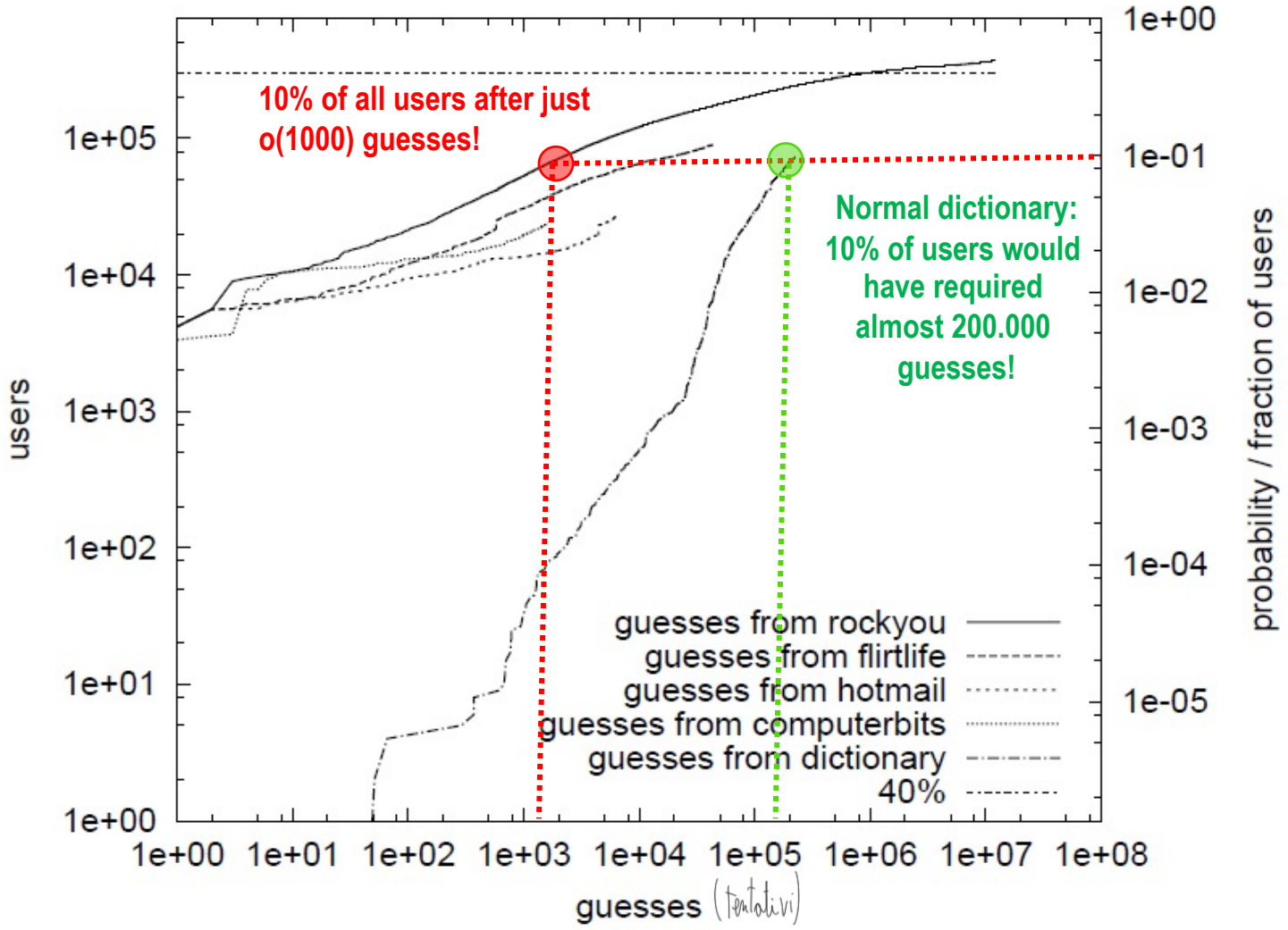


# More detailed stats

uso vecchie pw per attaccare nuovi dB

**source: David Malone, Kevin Maher, Investigating the Distribution of Password Choices, 2011**

**Use public passwd DBs to crack new ones (Gawker.com, released in 2010)**



# Dictionary rule-based attacks

➔ **SW specialized to «crack» a (protected, will see later) passwd database**

⇨ E.g., John the Ripper, hashcat, etc


➔ **Two main components**

⇨ A baseline database

⇨ Passwd generation rules

posso creare un generatore di password

| | | |
|---|---|---|
| Uppercase | → GIUSEPPE | |
| Capitalize | → Giuseppe | |
| Append/prepend | $1 | →giuseppe1 |
| | $abc | → giuseppeabc |
| | ^00 | → 00giuseppe |
| Replace | si1 | → g1useppe |
| | se3 | → gius3pp3 |

And many more, plus combinations of

Paradossalmente, siti che chiedono cambio pw sono meno sicuri di quelli che ti fanno usare la pw sempre. WHY?

▷ Quando la cambio, è spesso una rivisitazione della precedente (uso pw simili!) quindi con dipendenza.

• Come avere buon PW MANAGER? dato un 'secret' e stringa = sito internet (www.delphi.it) allora: HASH(secret, string-site). Solo con entrambi ricavo pw corretta.

# • Password-Based Authentication
# Vs
# • Challenge-Handshake Authentication

protocolli: **PAP, CHAP**

# Authentication: "proof of..."

## ➔To "prove" I know a password does NOT necessarily imply revealing it

- ⇨ Though revealing the password is ONE POSSIBLE approach
  - ➔We will see the PAP protocol later on – Passwd Auth Protocol

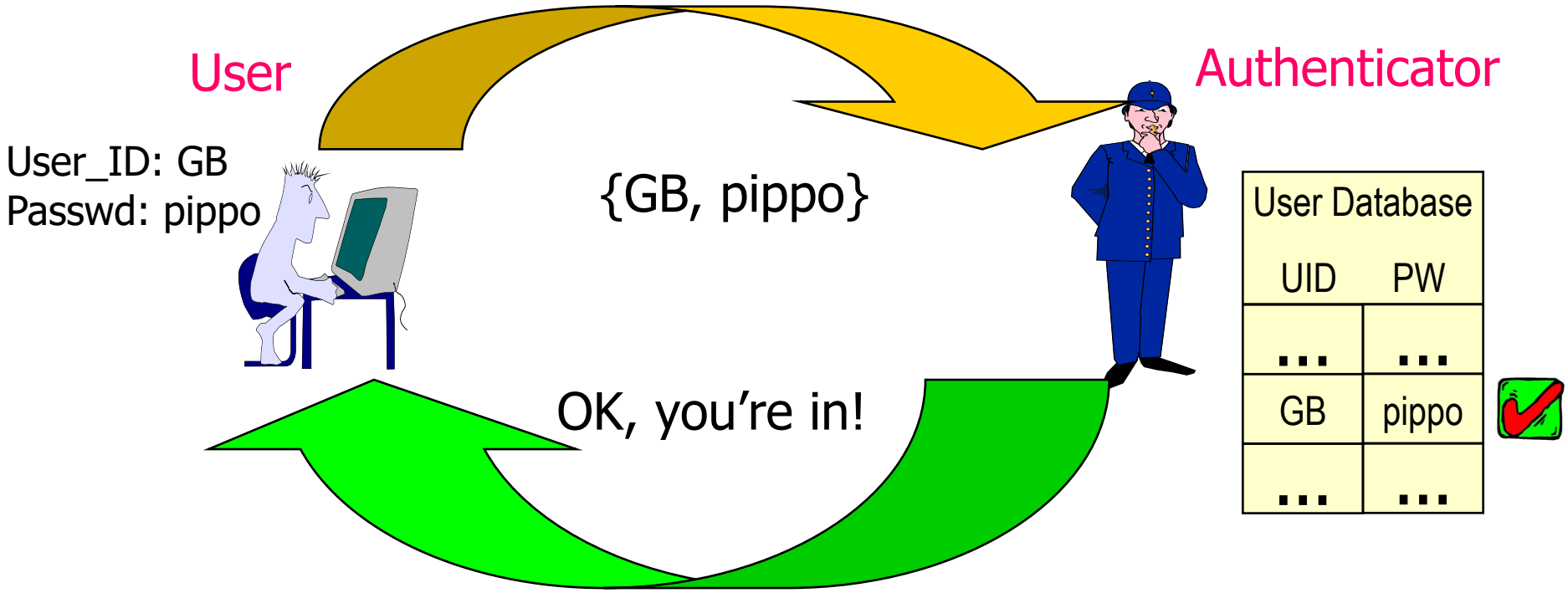
## ➔Different techniques → different levels of information disclosure

- ⇨ PAP = full disclosure
- ⇨ CHAP = some information leaks
- ⇨ ZKP = no information leakage

Zero Knowledge Protocols

# Password Authentication Protocol (PAP)
# Simplest possible auth approach

**Prove you know your password by… showing it in clear!**



# PAP obvious limitations

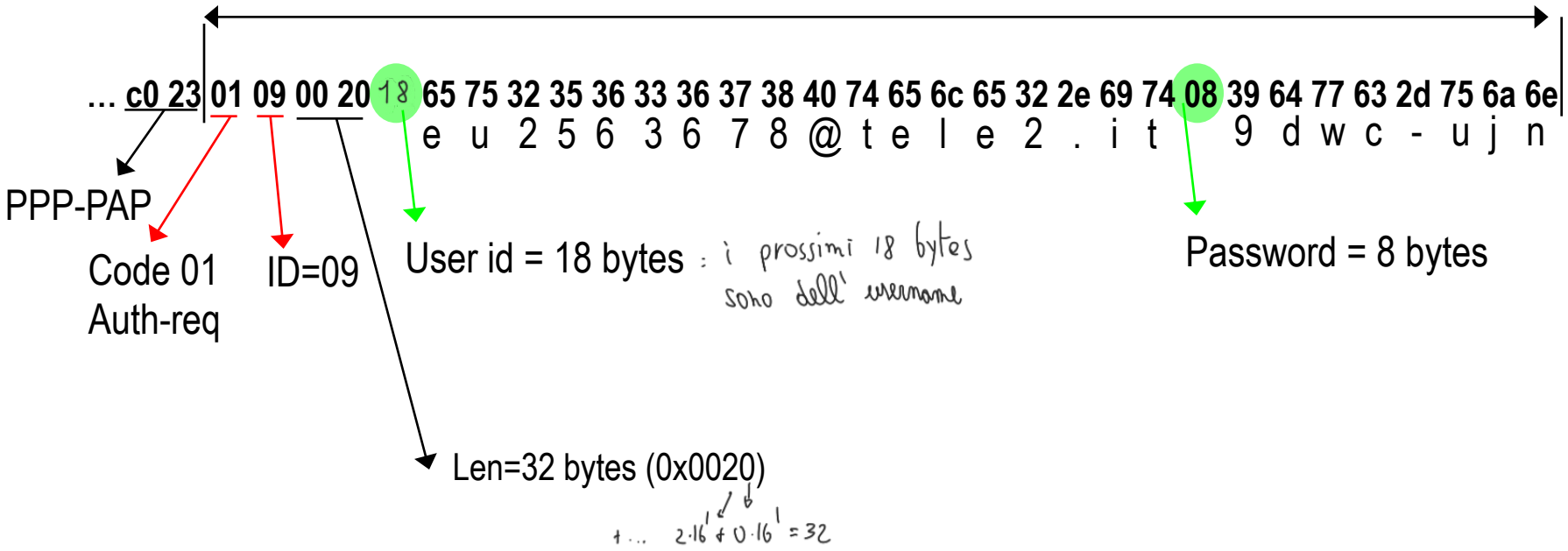
- **Passwords sent "in clear"**
  - If the channel permits eavesdropping (e.g. wireless networks), then game over

- **No protection from replay attacks**
  - You don't even need a "true" replay attack, since attacker already learns the password!

Se attacker tenta pw multiple, non viene bloccato, ∄ controllo!

- **No protection from repeated trial and error attacks**
  - Peer is in control of the frequency and timing of the attempts.

# PAP message example
## (real capture of a PPP trace)




ALL UID+PW IN CLEAR!!!!

è migliore? dipende da attacco.

# CHAP
# Challenge Handshake Authentication Protocol

# Authentication: “proof of…”

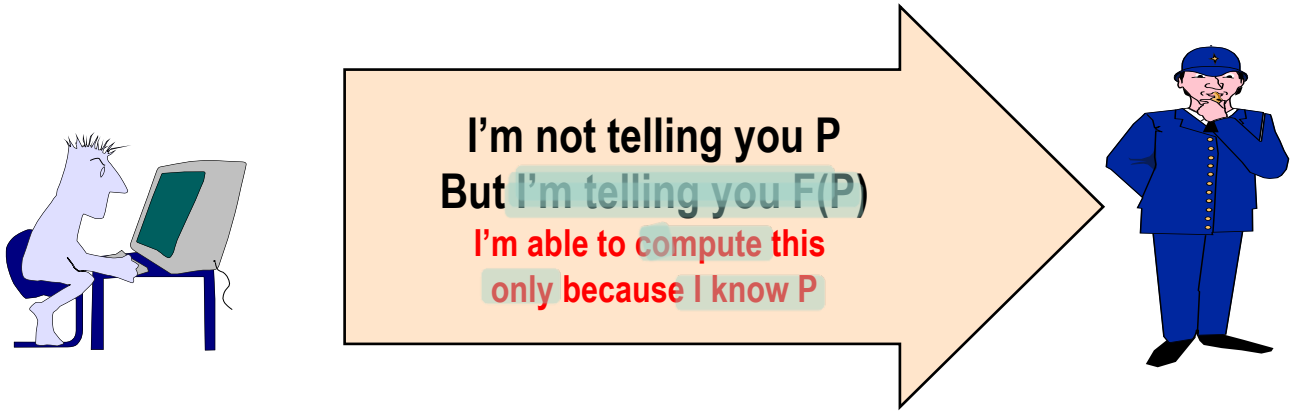
➔**Authentication: prove you know a secret or a password**

➔**PAP: prove you know the secret passwd by explicitly telling it to the authenticator**

- ⇨If attacker can hear this, game over
- ⇨If attacker can replay this, game over

➔**Are there alternatives? Yes!**

# Proof of knowledge: result of computation!




## ➔Which F(.)? Two properties

⇨Computation must NOT reveal the secret itself:

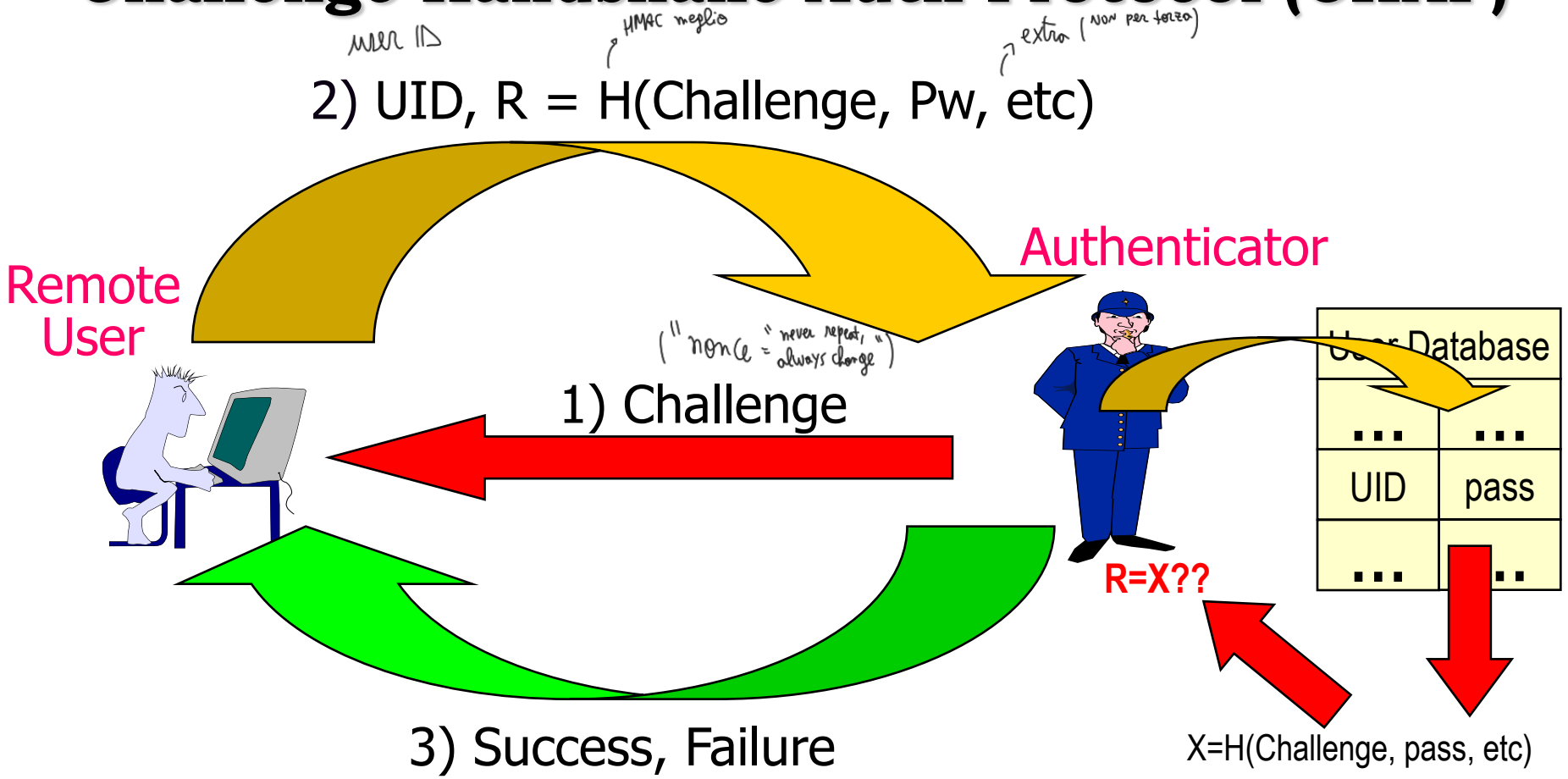
If res = F(P), P = $F^{-1}$(res) must NOT be computable…

» **We will very soon meet functions with such property ☺**

⇨F(P) must not be replayed by attacker! F cambia V accesso

F cannot be a function only of P, but must include a **nonce**

# Challenge-Handshake Auth Protocol (CHAP)




➔ **Secret passwd never transmitted in clear**

- ⇨ Secure against adversary that can eavesdrop the channel
- ⇨ Hash is the usual approach, but it is NOT the only one
  - ➔ Other common approach: encrypt challenge using P as secret key. But requires more care!

# CHAP pros & cons

**Pros:**

➔ **Protection against playback (replay) attack**

→ But make sure challenge NEVER repeats!

➔ **Repeated challenges**

⇨ Authentication may be repeated over connection time (unlike PAP where it is performed only once at start)

→ Authenticator controls frequency and timing of the challenges

⇨ intended to limit the time of exposure to any single attack

➔ **Initiated by Authenticator**

⇨ Peer cannot initiate, as challenge must first be sent

**Cons:**

➔ **Secret must be available in plaintext form**

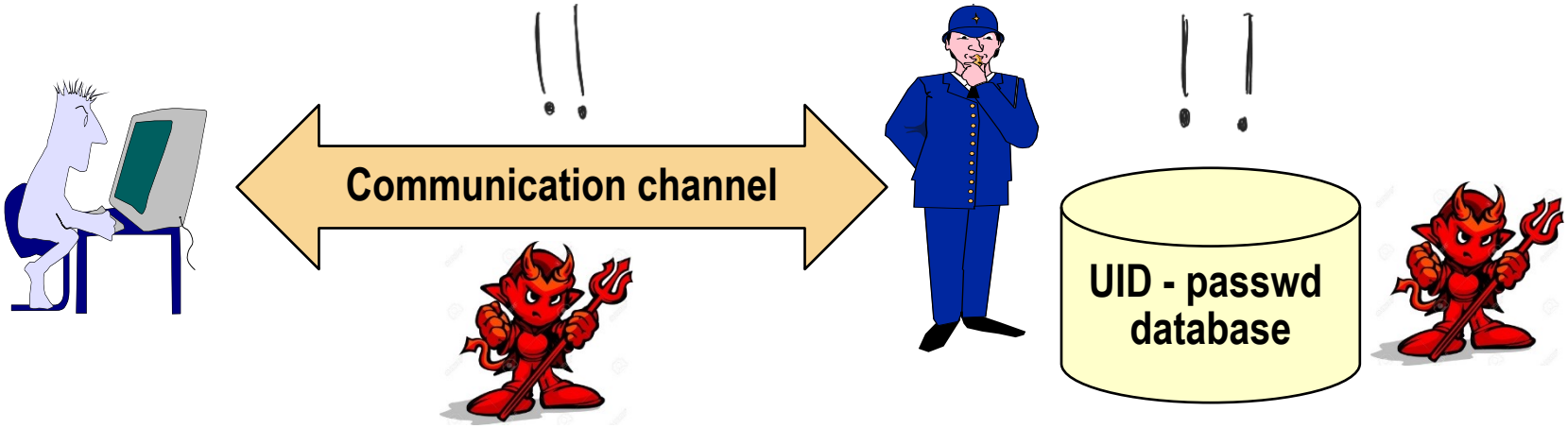
⇨ Cannot use irreversibly encrypted password databases

⇨ More later on this

# So, PAP vs CHAP: which one is better then?

**(less obvious than what you might think at this stage!!!)**

# 2 main (*) attack models




➔ **Attack to the communication channel**

- ⇨ Eavesdropper, MITM, replay, …
- ⇨ Game Over!
  - →If you want to use PAP, you MUST protect communication channel
  - →E.g. online login via https, EAP-TTLS, etc

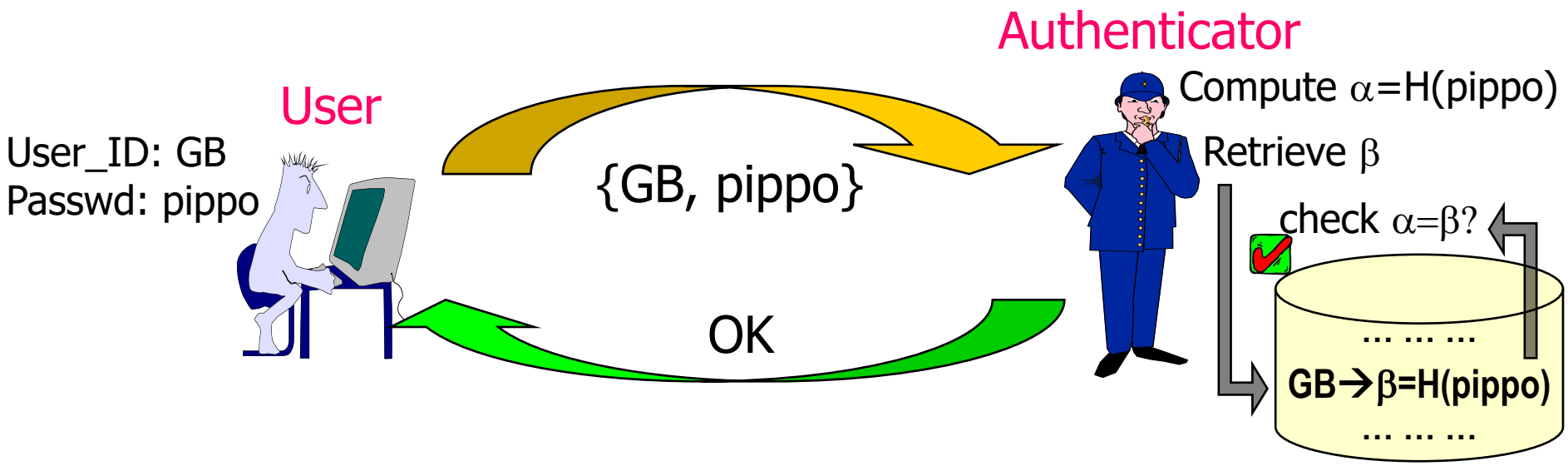
➔ **Attack to the backend UID-passwd database**

- ⇨ Attacker penetrates system and steals entire passwd DB
- ⇨ **Mitigation: hashed password database!**

**(*) other system-level attacks: insiders, workstation hijacking, key loggers, fake screens, etc**

# Hashed passwd database in PAP

➔**Idea: store H(passwd) instead of passwd**

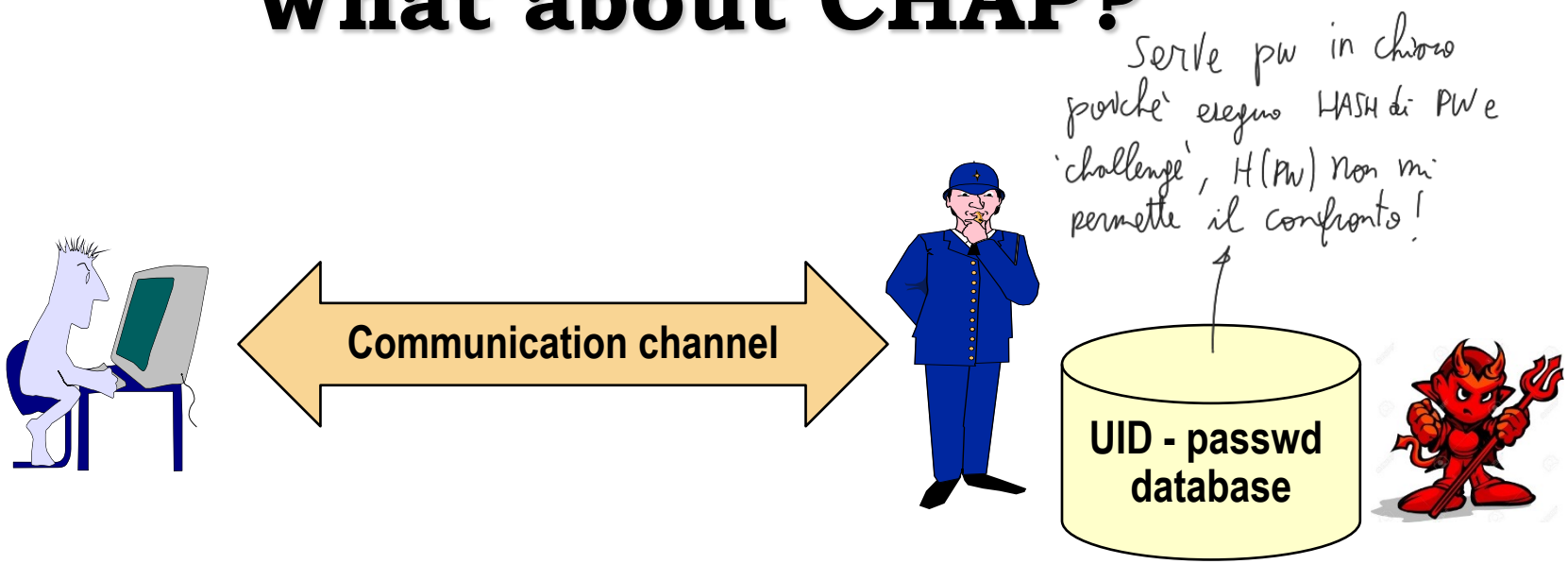



➔**Improved security: if attacker steals DB, still has to «invert» passwd**

⇨Good hash = one-way → only method is brute-force/guess

→Security ultimately depends on choosing «strong» passwd

# What about CHAP?




➔**Attack to the backend UID-passwd database**

⇨CHAP: cannot anymore hash passwd!!!

➔**Conclusion**

⇨If Attacker = eavesdropper, then CHAP better

⇨But if Attacker = backend, then PAP better!

➔**Hence there is not a «single» solution, but all depends on which attack you aim at defending**

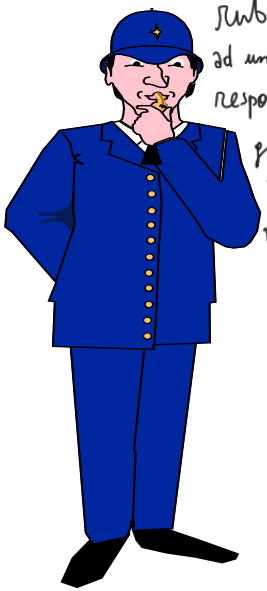
# Why no hashed pw in CHAP?

e se Response = H( H(pw) | challenge ) ? nel DB passo avere H(pw). Mi immedesimo nell'hacker: rubo DB, vedo user:GB pw: H(pw), ad un certo punto cambia la challenge, response = H(H(pw) | newCh), ma ho già H(pw), è un'illusione, ho solo perso pw ≐ H(pw), ma se poi uso H(pw) e non pw è inutile !!

CHALLENGE = 135623

RESPONSE = Flavia, H(mypass | 135623)

ACK or NACK



| User Database | |
|---|---|
| ... | ... |
| flavia | mypass |
| ... | ... |

- ➔ **No way to compute H(pw, challenge) without having pw in clear!**
- ➔ **Consequence: user DB must remain in clear → straightforward target for attack!**

- ➔ **Conclusion: CHAP is worse than PAP against a backend attack model!!** DB deve essere ben protetto!

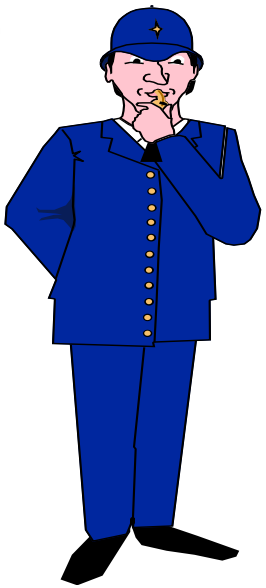
Normalmente si usa PAP on top of HTTPS (≐ TLS) per proteggere il canale.

# Mitigation: (explicit) "salt"

CHALLENGE = 135623; SALT = 9876

RESPONSE = Flavia, H(H(9876,mypass) | 135623)

ACK or NACK



| User Database: SALT=9876 | |
|---|---|
| … | … |
| flavia | H(9876,mypass) |
| … | … |

➔ **Same idea as previous salt**

- ⇨ Salt value is in clear
- ⇨ Store H(salt, passwd)

➔ **very different reason and use!**

- ⇨ Attacker cannot anymore «reuse» stolen passwd!
  - ➔ **But of course passwd strength and dictionary attacks still hold…**
- ⇨ After a breach (or periodically), regenerate entire DB using new salt

# • Quale funzione Hash usare?

- MD5? NO, SHA256? è adeguata? Perché non dovrebbe?
  SHA usato per H(secret|msg), qui H(msg), che cambia?

• qual è il tipo di attacco? rubare pw da dB e ripete Hash
  (fast 'crack')
  SHA è "veloce", ma causa bitcoin 7 MW per computarlo!

- i7 3930K → 66.6 MH/s
- Antminer → 67 TH/s

} x 67 volte

esempio: 8 bytes random string con 64 caratteri

allora $64^8$ segreti differenti (NO PW) → 2,5 s per forzarlo.

Serve hash 'lenta' come Bcrypt (è "regolabile" la velocità)

trovare sempre l'elemento ADATTO, un elemento è migliore in un campo, ma peggiore in un altro!!

• Ma attacco in cui è possibile ripetere tentativi senza 'blocchi' NON USO SHA256 (causa BITCOIN), allora dovrei toglierlo!?

ESEMPIO:

uso paradox Compleanno, rompo sistema se trovo collisione:

128 bit → $2^{128}$ values $\approx 34 \cdot 10^{37}$ → $\frac{2^{128}}{110 \cdot 10^{12}\ \text{TH/s} \simeq 10^{14}\ \text{H/s}}$ $\left[\frac{\text{tentativi}}{\text{second per trovare collisione}}\right] = 3{,}4 \cdot 10^{24}$ s / 60 m / 60 h / 24 d / 365 = $1{,}07 \cdot 10^{17}$

(1 TH/s = $1 \cdot 10^{12}$ H/s)

Contro collisione ho un tempo non indifferente, non considerabile "rotto"!

CONTRO COLLISIONI OK (prodotto da dipert), nella PW "esplora" possibili input, con spazio pw molto ridotto.