Lecture 2:

Basic
PPP authentication mechanisms

PAP, CHAP, +++

Recommended reading:
RFC 1334, October 1992;
RFC 1994, August 1996
Wiley AAA book, chapter 2 (parts)

Authentication in PPP

➔**Optional phase**

⇨After link establishment

⇨after or during link quality determination (if present)

➔**Authentication mechanism negotiated during link establishment**

⇨Option sent in configuration request PPP msg

Basic PPP authentication

➔ **LCP option #3 = authentication protocol**

Type 1 byte 03	Length 1 byte >=4	Auth-protocol 2 byte c023=PAP, c223=CHAP	data 0+ bytes …

Specific extra info needed to the considered auth protocol

➔ **Two basic authentication mechanisms initially considered**

⇨ PAP: Password Authentication Protocol

Type 1 byte 03	Length 1 byte 04	Auth-protocol 2 byte c023=PAP

No extra info

⇨ CHAP: Challenge Handshake Authentication Protocol

Type 1 byte 03	Length 1 byte 05	Auth-protocol 2 byte c223=CHAP	data 1 byte 05=MD5

Extra info: Hash Function
Basic = MD5

Auth direction

➔ **Independently done on both directions!**

⇨ Authentication may differ

⇨ or may apply to a single direction only

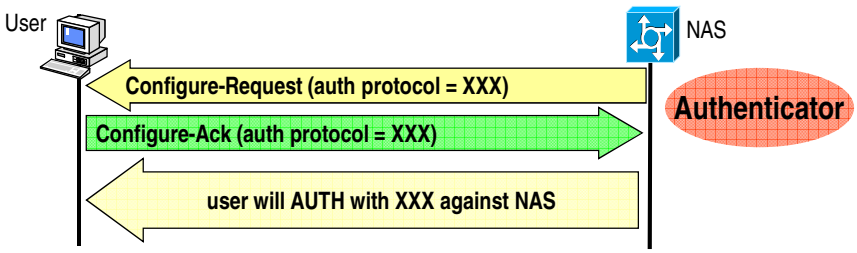
➔Typically NAS requires user to authenticate

➔User does not require NAS to authenticate

➔ **Authenticator = the end of the link that requires the other peer to perform authentication**

➔ **Authenticator: sends the Configure-Request, specifying the authentication protocol to be used**

⇨ Both sides act in turn as authenticators in the case of mutual authentication



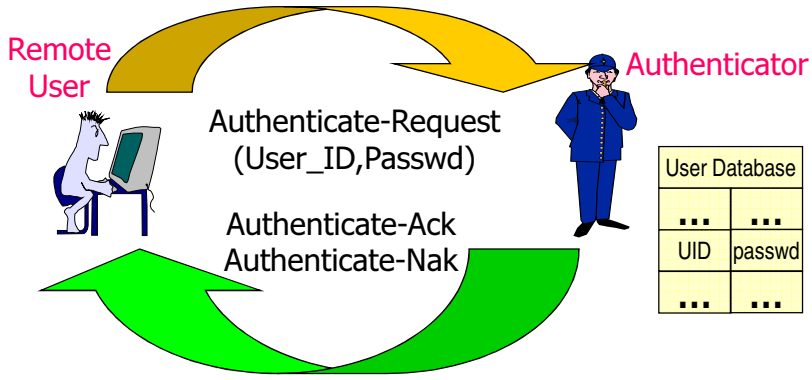
PAP
Password Authentication Protocol

Password Authentication Protocol

➔ **Simplest possible mechanism**

⇨ Based on the a-priori knowledge, by the authenticator, of the (user_id, password) pair specified during contractual agreement

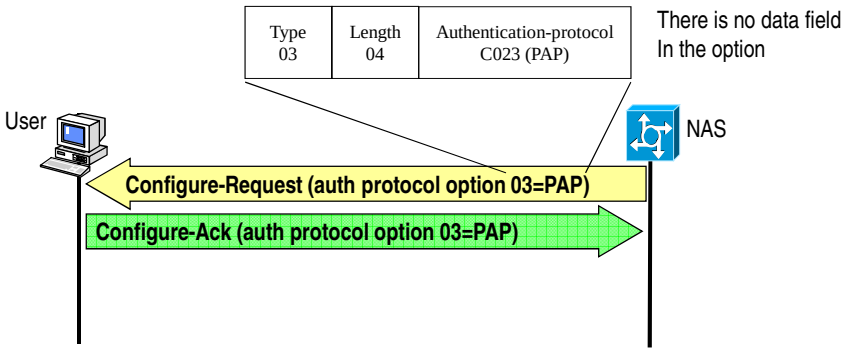
➔ **Two-way handshake**



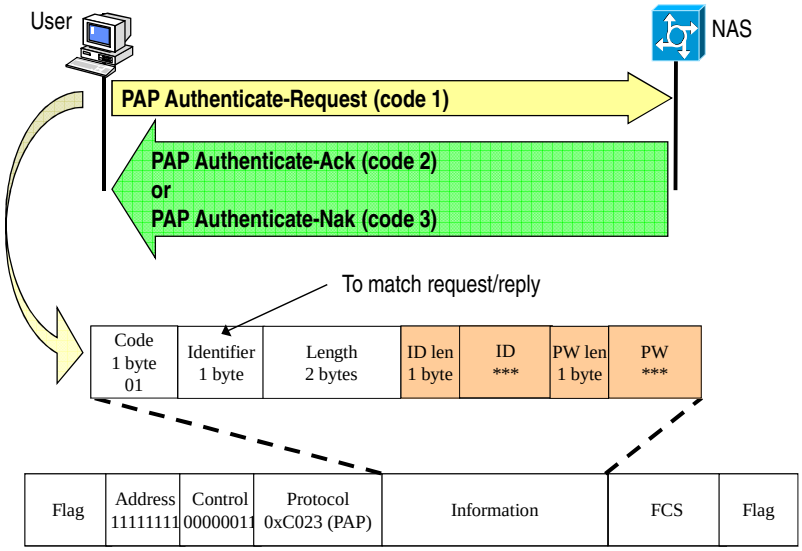
PAP procedure

- **Starts when link is established**
- **Authenticating peer repeatedly sends Id/Password pair, until:**
  - An ACK is received
  - A NACK is received and/or the connection is terminated
- **PAP is a weak authentication method**
  - Passwords sent "in clear"
  - No protection from playback
  - No protection from repeated trial and error attacks
    - Peer is in control of the frequency and timing of the attempts.

PAP Authentication Protocol option



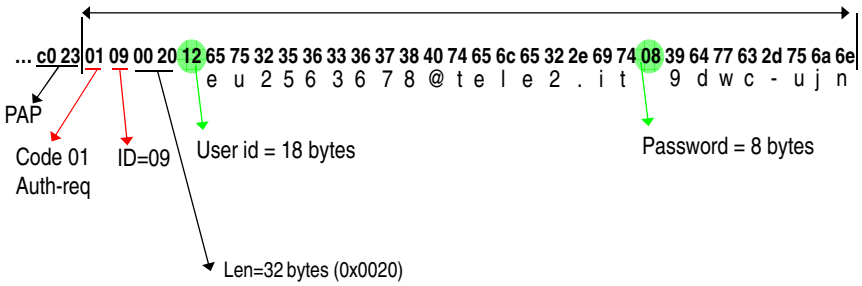
PAP packet format: auth-request



Giuseppe Bianchi

Auth-request example

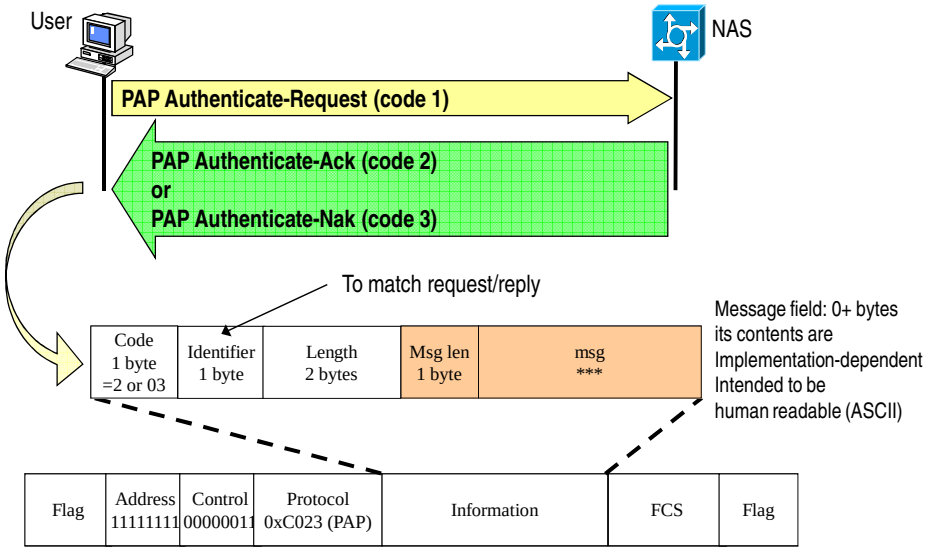
(real capture)



ALL UID+PW IN CLEAR!!!!

Giuseppe Bianchi

PAP packet format: auth-Ack/Nak



Giuseppe Bianchi

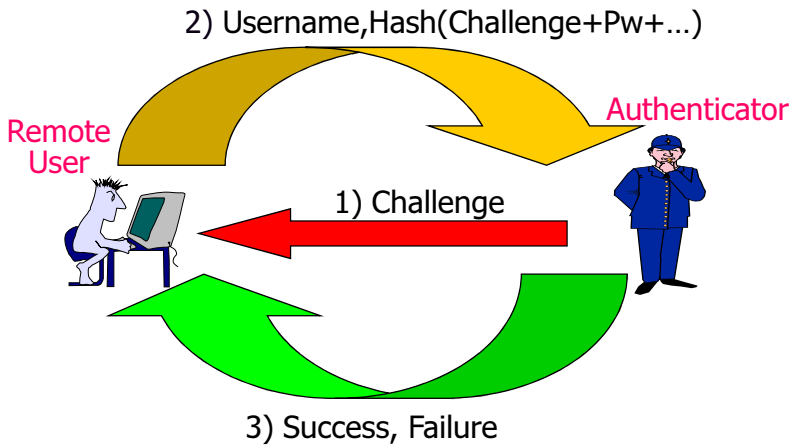
CHAP
Challenge Handshake Authentication Protocol

Giuseppe Bianchi

Approach

- **Three-way handshake**
  - Challenge – Response – Success or Failure
- **Uses a Random Challenge, with a cryptographically hashed Response which depends upon the Challenge and a secret key**
- **Secret key never transmitted in clear**
  - Much more safer than PAP
- **Conceptually identical to the approach currently adopted into actual cellular networks**
  - GSM, UMTS

Three-way handshake



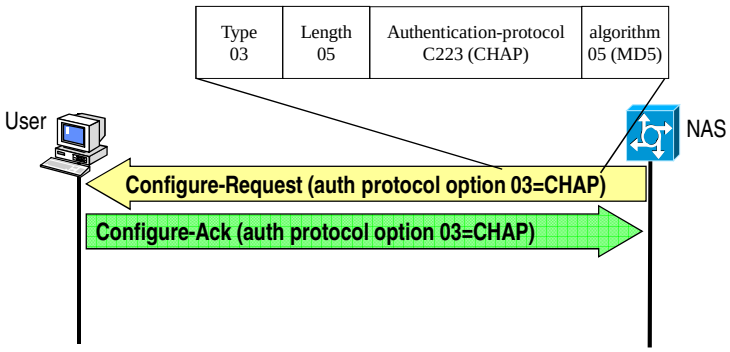
Three-way handshake performed initially, after link establishment
But it MAY be repeated ANYTIME at RANDOM TIMES after the link is established
With new (different) challenges!!

CHAP pros & cons

- **Pros:**
  - Protection against playback attack
    - Incrementally changing identifier
    - Variable challenge value
  - Repeated challenges
    - Authentication may be repeated over connection time (unlike PAP where it is performed only once at start)
    - intended to limit the time of exposure to any single attack
  - Authenticator controls frequency and timing of the challenges
    - CHAP does not allow a peer to attempt authentication without a challenge
- **Cons:**
  - Secret must be available in plaintext form
    - Cannot use irreversibly encrypted password databases
  - Hard scalability in large installations
    - every possible secret must be maintained at both ends of the link

CHAP selection

Similar to PAP: during configure-Request



The only required hashing algorithm, in a conforming implementation, is MD5 (but CHAP protocol open to other possible hashing algorithms as well)

CHAP Challenge

- **Identifier: MUST change at each new challenge**
- **Value: randomly generated - must be designed to be**
  - Unique & different per each challenge
    - → To avoid replay attacks
  - Not predictable
  - Value size: 1+ bytes
    - → In principle arbitrary and independent of the hashing algorithm used
- **Name field: identification of the system transmitting the packet**

Code 1 byte 01	Identifier 1 byte	Length 2 bytes	Ch.Len 1 byte	Challenge ***	Name ***

```
PPP Challenge Handshake Authentication Protocol (REAL TRACE EXAMPLE)
    Code:          0x 01 (Challenge)
    Identifier:    0x 01
    Length:        0x 00 1f (31 bytes)
    Value Size:    0x 10 (16 bytes)
    Value:         0x 07 21 c9 b3 30 a6 f8 6f 52 ff 67 7f 07 3d 15 f5
    Name:          MILZ-LNS-9 (10 bytes)
```


Response computation

- **One-Way Hash function**
  - Transform an arbitrary text size into an alphanumeric sequence of given size (digest)
- **MD5 digest = 16 bytes**
- **Response value: one-way hash calculated over:**
  - Identifier, concatenated with the "secret", concatenated with the challenge value

"secret" size:
- → At least 1 byte
- → Typically more (a "normal" password)
- → Preferable: at least 16 bytes (MD5 digest size)

Original text — **One-way Hash** — **Digest**

Code 1 byte 02	Identifier 1 byte	Length 2 bytes	Val len 1 byte	Value ***	Name ***

```
PPP Challenge Handshake Authentication Protocol (REAL TRACE EXAMPLE)
    Code:          0x 02 (Challenge)
    Identifier:    0x 01
    Length:        0x 00 27 (39 bytes)
    Value Size:    0x 10 (16 bytes)
    Value:         0x 4b 70 76 c3 2a b5 21 f0 29 9b 25 72 06 0a e4 ae
    Name:          eu2563678@tele2.it  (18 bytes)
```

Success/Failure

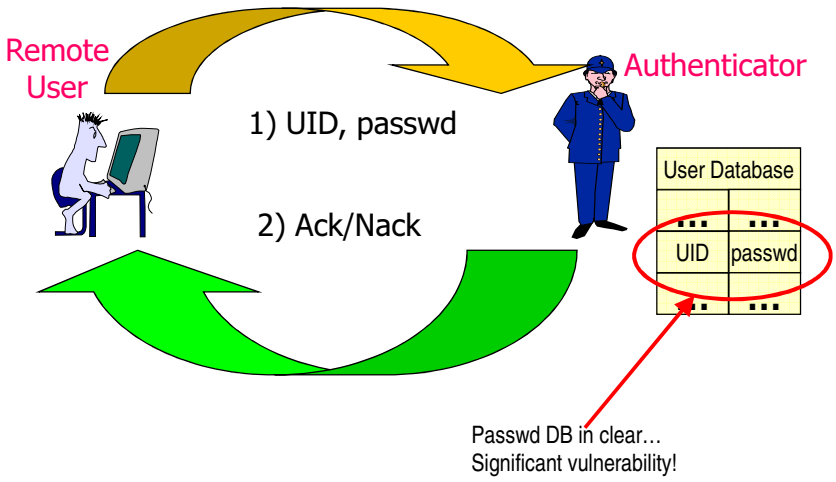
- **Authenticator in turn computes the digest**
  - It has identifier, challenge, and the secret key
    - → In fact there is the user id repeated in the "name" field (password from DB lookup)
- **And compares with that received**
  - If OK, send back Success (Code 03)
  - If NO, send back Failure (Code 04) and terminate link
- **Message: optional field, intended for human**

Code 1 byte 03 or 04	Identifier 1 byte	Length 2 bytes	Message ***

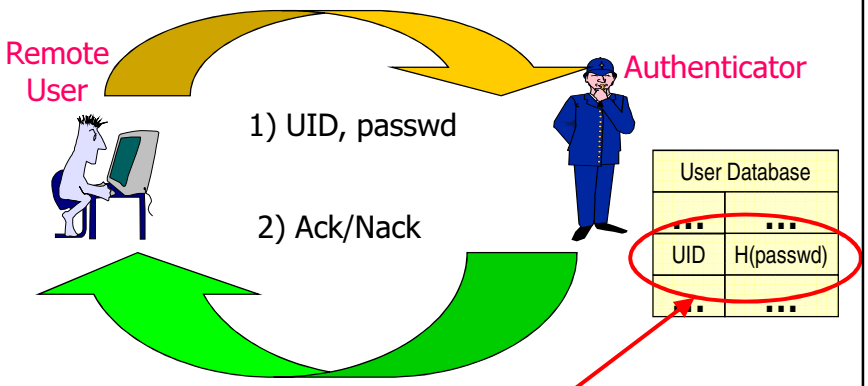
```
PPP Challenge Handshake Authentication Protocol (REAL TRACE EXAMPLE)
    Code:          0x 03 (Challenge)
    Identifier:    0x 01
    Length:        0x 00 04 (4 bytes)

[no message in the considered inmplementation!]
```


Password based Authentication: Extensions

Passwd protection in DB?



Passwd protection in DB: storing passwd hashes!



Authenticator:

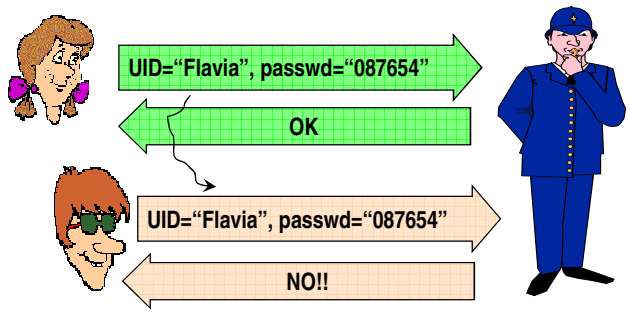
1) receives UID & passwd (clear text)
2) computes hash H(passwd) - any locally used Hash OK; Linux = MD5
3) compares with DB entry

Dictionary attack…

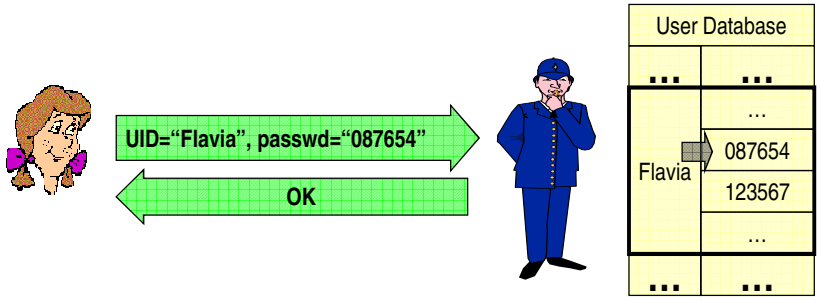
- ➔**Many users use predictable passwd**
- ➔**Dictionary attack:**
  - ⇨Hashing does not help
  - ⇨Will see in a dedicated laboratory!

One-time passwd

- ➔**Is it possible to extend PAP so that user changes passwd at every (successful) attempt?**
  - ⇨If it is, would prevent playback attacks

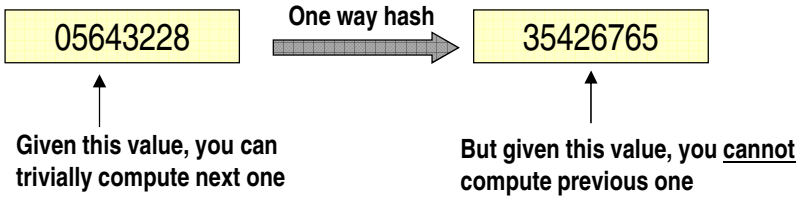


One-time passwd: trivial… but…



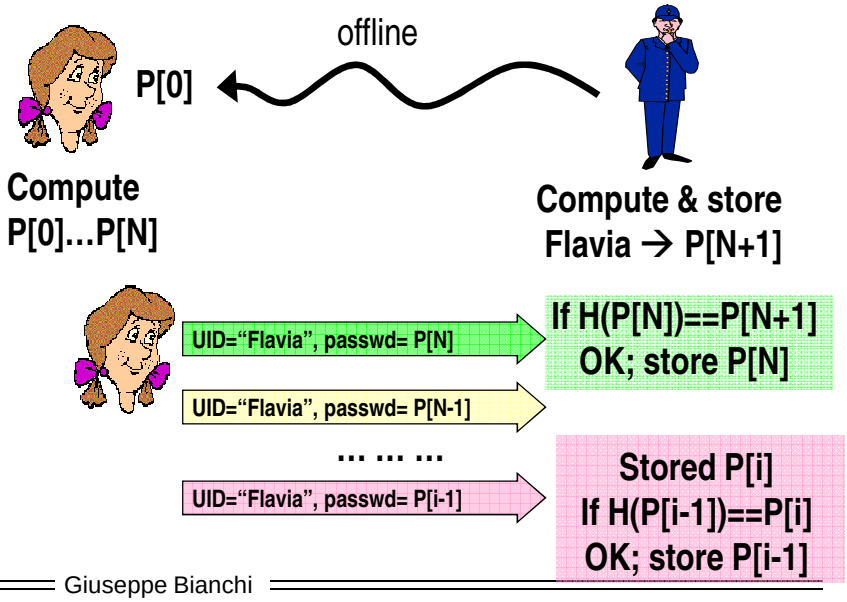
➔N (large) passwd per user
➔10.000.000++ users
➔HUGE DB!! Not viable

Idea: hash chains



P[0] = starting point
P[i] = H(P[i-1])
P[N] = last value

One-time passwd: practical



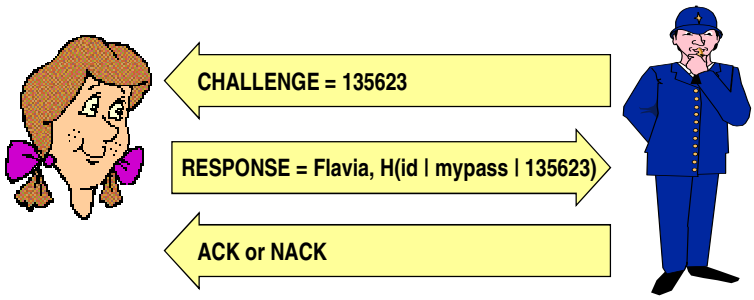
One-time passwd benefits

- **Passwd in clear = OK**
- **Authenticator only stores USED passwd**
  - no way to predict next one (1-way hash)
- **Authenticator only stores 1 value**
  - Same complexity as in ordinary PAP

- **Issues:**
  - Large N to prevent frequent renegotiation
  - Client size = vulnerable (must store passwd seed or whole vector)

Issues with CHAP

Basic CHAP vulnerability

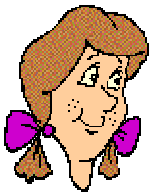


User Database	
...	...
flavia	mypass
...	...

➔ **Authenticator MUST store passwd in clear!**
- ⇨ Otherways no way to compute H(id, pw, challenge)

➔ **Authenticator storage = straightforward target for attack!**
- ⇨ Even worse than PAP!!

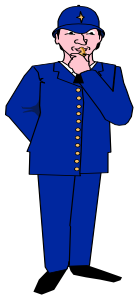
Idea: "salt"



CHALLENGE = 135623; SALT = 9876

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

ACK or NACK



➔**Attacker may only access to "salted" passwd**

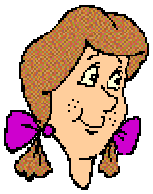
- ⇨Different salt for different authenticator servers
  - ➔breaking one != breaking all
- ⇨Refresh DB periodically
  - ➔Someone must take care of this… more later

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

SALT: Same idea can be used also in PAP (of course)

Giuseppe Bianchi

CHAP and mutual authentication /1



ID=2; CHALLENGE = 135623

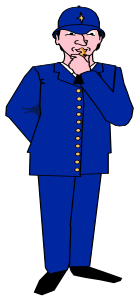
RESP: Flavia, H(ID=2, sharedsecret, 135623)

ACK

ID=3; CHALLENGE = 324567

RESP: servername, H(ID=3, sharedsecret, 324567)

ACK

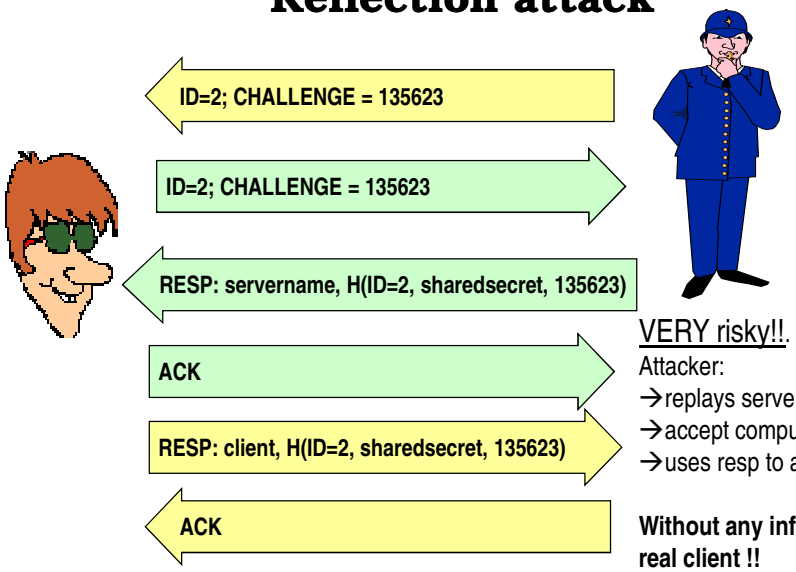


Usage of a shared Secret… good idea, Easy to manage!

Good idea??

Giuseppe Bianchi

CHAP and mutual authentication /2
Reflection attack



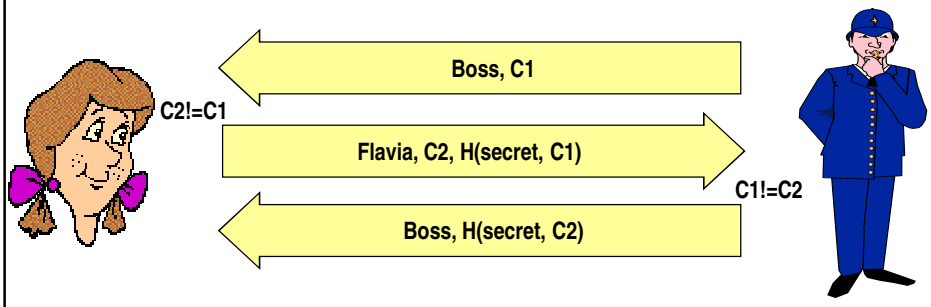
VERY risky!!.
Attacker:
→replays server challenge
→accept computed resp
→uses resp to authenticate!!

Without any info on real client !!

Mutual authentication with Challenge-Response

(just some hints… no full analysis)

Basic idea

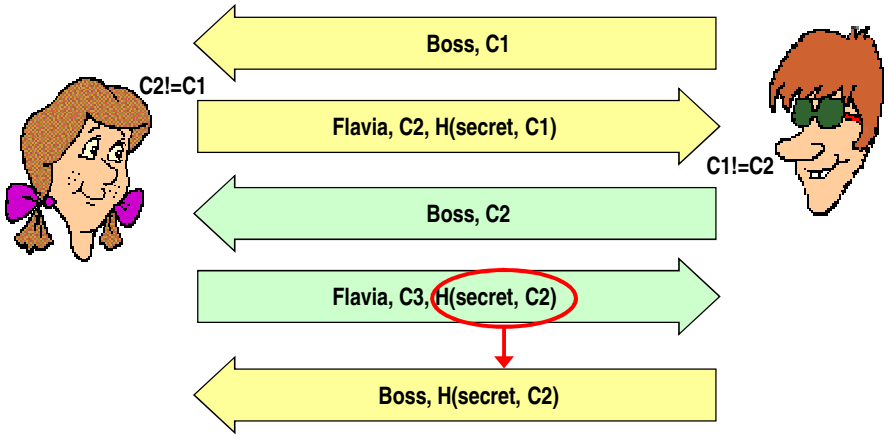


Flavia shows knowledge of secret over C1

Boss shows knowledge of secret over C2

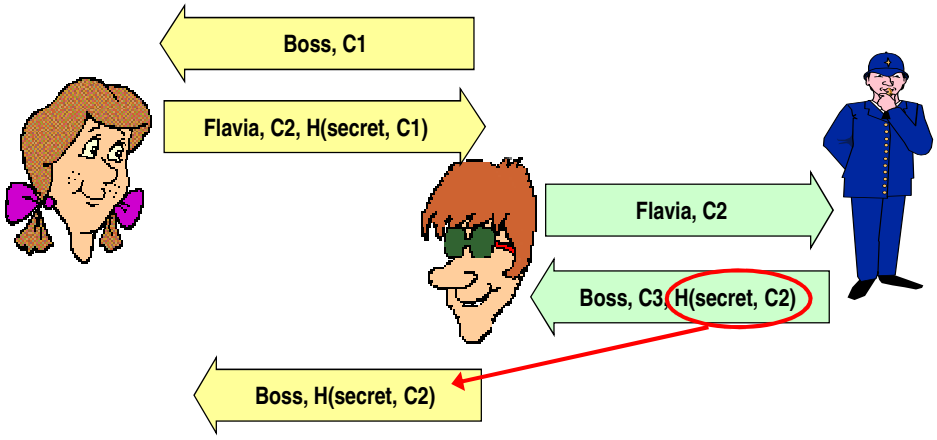
Reflection!

➔**Does not work**

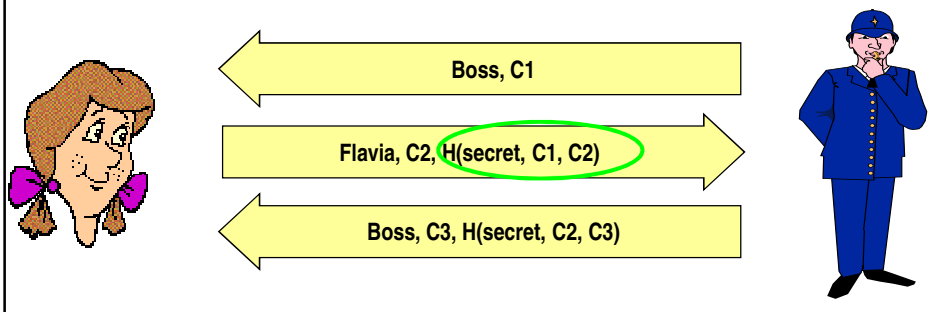


What if reflection is prevented by protocol status?

➔Attacker may use "other" party!

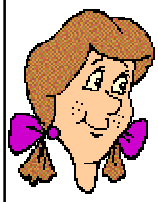


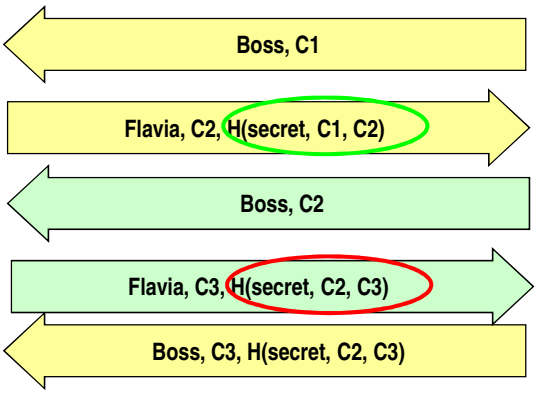
Let's try to fix this



Chaining challenges! Add dependency between challenges in same handshake

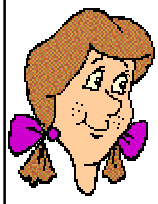
Does not work

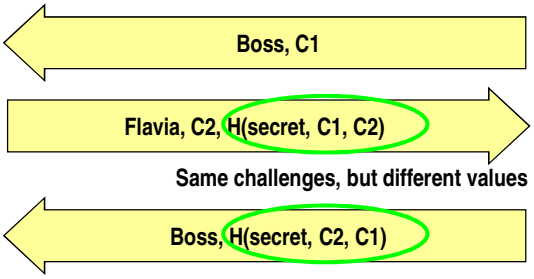




Too many nonces!!

Minimize nonces

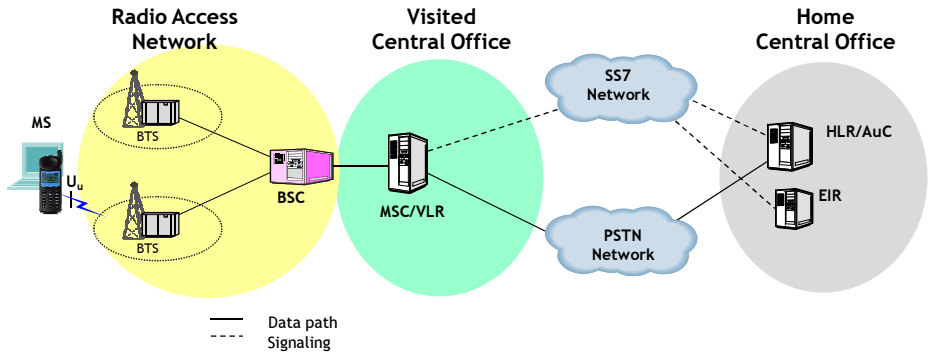




No reflection possible anymore

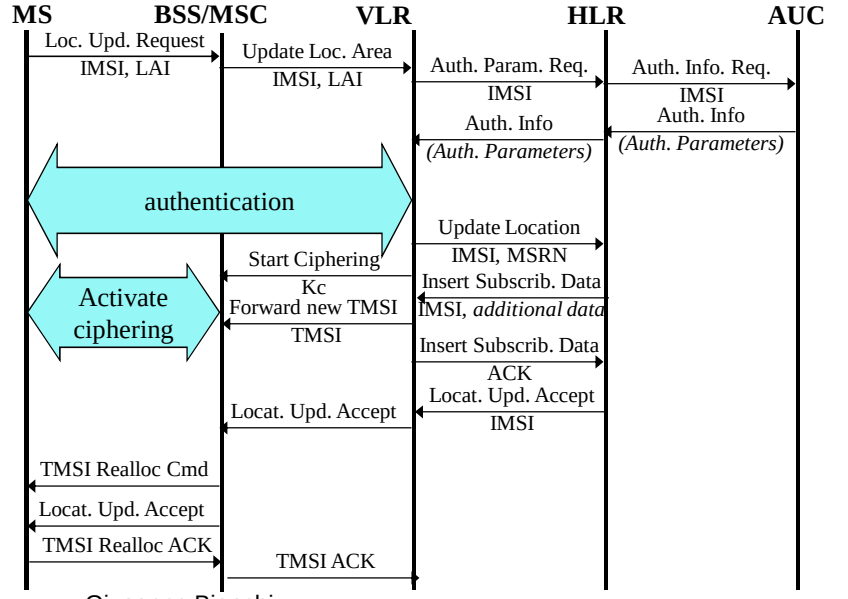
Challenge-Response in GSM authentication

GSM essential components

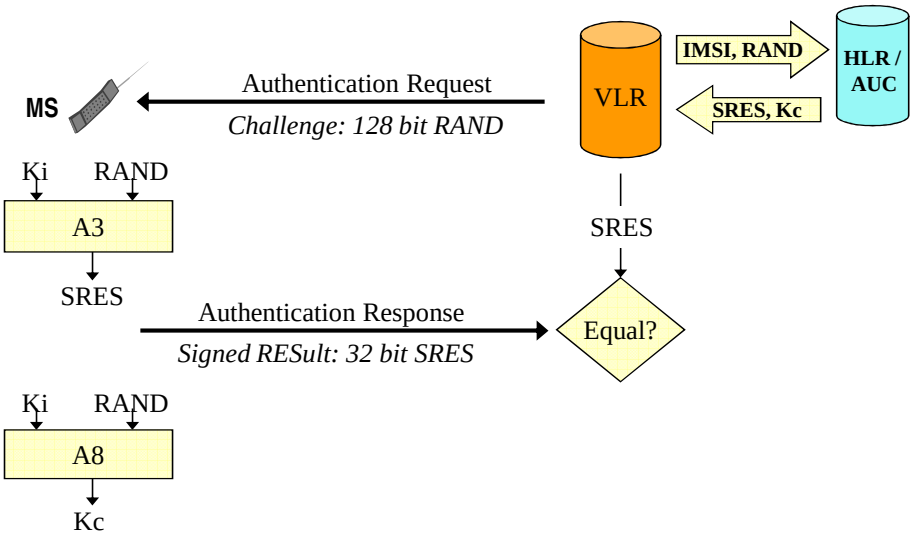


- MS: **Mobile System**
- HLR: **Home Location Register**
- VLR: **Visiting Location Register**
- AuC: **Authentication Center**
- BTS: **Base Transceiver Station**
- BSC: **Base Station Controller**
- SIM: **Subscriber Identity Module**
- MSC: **Mobile Switching Center**
- EIR: **Equipment Identity Register**

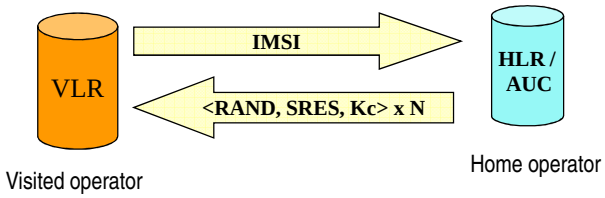
Authentication: when



Authentication
(managed by VLR)



Triplets (Authentication Vector)

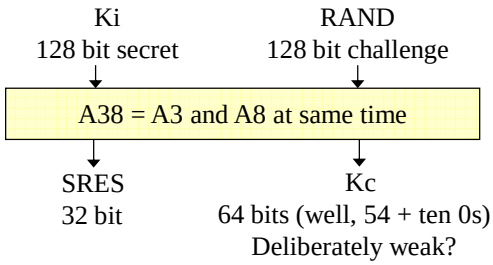


➔ **Idea: once in a VLR area, autentication will need to be performed MANY times**

➔ **Hence deliver N triplets, to be used for N distinct authentications**

➔ **IMPORTANT: VLR does NOT need to know authentication algo used (A3, A8)**
- ⇨ Triplet contains computed result by AuC
- ⇨ A3, A8 run inside the SIM (given by operator)

Authentication: details



➔**Challenge response with:**
- ⇨Challenge → RAND
- ⇨Secret → Ki
- ⇨Hash → A3 algorithm

On the A3/A8 algorithms

- **Security by obscurity**
  - A3 algorithm CAN BE operator-specific!
  - But most vendors originally used algo today called COMP128
  - Non disclosed but…
    - Reverse engineered? Leaked out?
- **COMP128 broken in 1998**
    - http://www.isaac.cs.berkeley.edu/isaac/gsm-faq.html
  - Chosen challenge attack [1998, Briceno, Goldberg, Wagner]
    - Ki disclosed through about 150.000 queries with suitably selected challenges → approx 8h in 1998
    - Having Ki means cloning the card!
  - Better attack [2002, Rao, Rohatgi, Scherzer, Tinguely]
    - Less than 1 minute!
- **Lesson learned:**
  - Security by obscurity does NOT work!! Leave hash design to crypto experts

Over the air attach

- **No mutual authentication!**
- **Rogue BTS may easily perform the attach**
  - run it for sufficient time

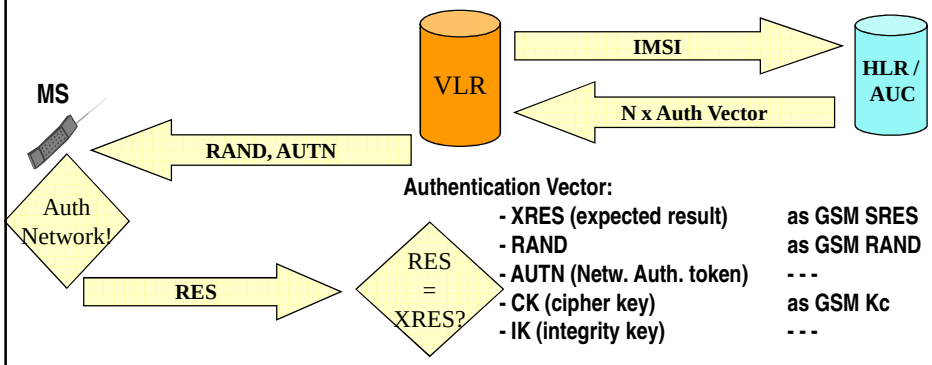
UMTS authentication: AKA Authentication and Key Agreement

(Simplified for our purposes)

Major differences with GSM

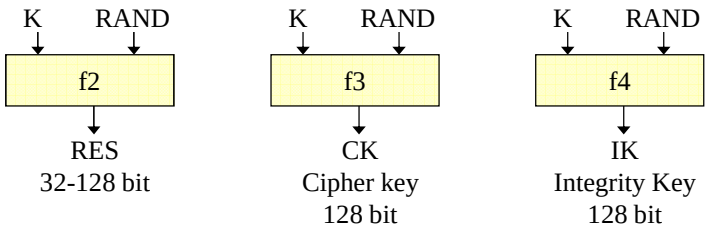
- **Mutual authentication!**
  - Optimized…
  - Guaranteed freshness for auth parameters
- **More comprehensive security**
  - More keys and several extra details
    - (we will not focus on this)
- **Algorithms FIRST scrutinized by research community, THEN selected**
  - The opposite of security by obscurity ☺

Authentication Vector



MS authentication

➔Usual challenge response, but different (public) algorithm



Network Authentication

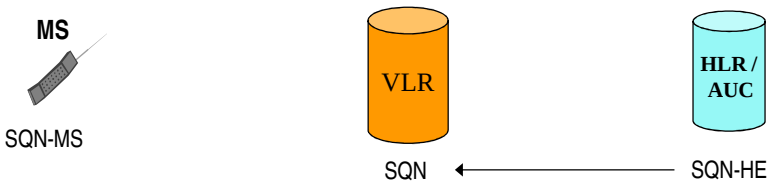
➔**MS should send a nonce…**

⇨1 extra message

➔**Bright idea: use Sequence number as "implicit" nonce!**

⇨Issue: MS and AuC must be (approx) synchronized

⇨And robust procedures for resync must be specified



SEQ as nonce: idea

Current SQN-MS stored

Check SQN = SQN-MS+1 (or in appropriate Tolerance range to come with lost msg)

SQN (included in AUTN) ←

Use SQN as "implicit" challenge!!

Once auth OK, update local SQN

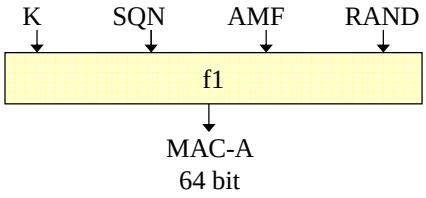
Network authentication: AUTN format

SQN Sequence number	AMF Auth & key mgmt field	MAC-A Message Auth code

48 bit

16 bit: carries info on which algo or key to use if choice abailable (signalling info)

64 bit: allows MS to verify authenticity of Network!



MS: has all info needed to check that MAC-A transmitted by network is the same of MAC-A locally computed! SQN guarantees defense against replay

Minor detail: protecting SQN!

➔A privacy problem:

⇨By looking at SQN (stepwise increasing), eavesdropper may discriminate and track user!

➔Solution: mask SQN with Anonymity Key

SQN xor AK Sequence number	AMF Auth & key mgmt field	MAC-A Message Auth code