

Lecture 2:

**Basic
PPP authentication mechanisms**

PAP, CHAP, +++

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

===== Giuseppe Bianchi =====

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

===== Giuseppe Bianchi =====

Basic PPP authentication

→ LCP option #3 = authentication protocol

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

Specific extra info needed to the considered auth protocol

→ Two basic authentication mechanisms initially considered

⇒ PAP: Password Authentication Protocol

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

No extra info

⇒ CHAP: Challenge Handshake Authentication Protocol

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

Extra info: Hash Function
Basic = MD5

===== Giuseppe Bianchi =====

Auth direction

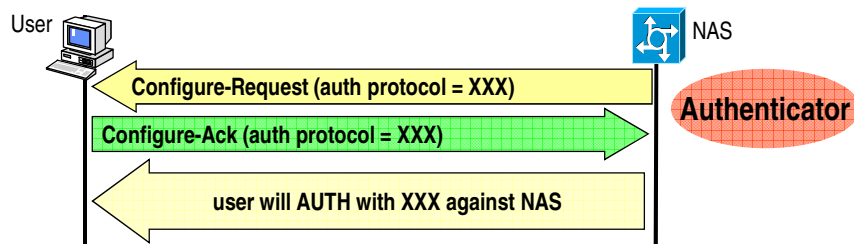
→ Independently done on both directions!

- ⇒ 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



===== Giuseppe Bianchi =====

PAP

Password Authentication Protocol

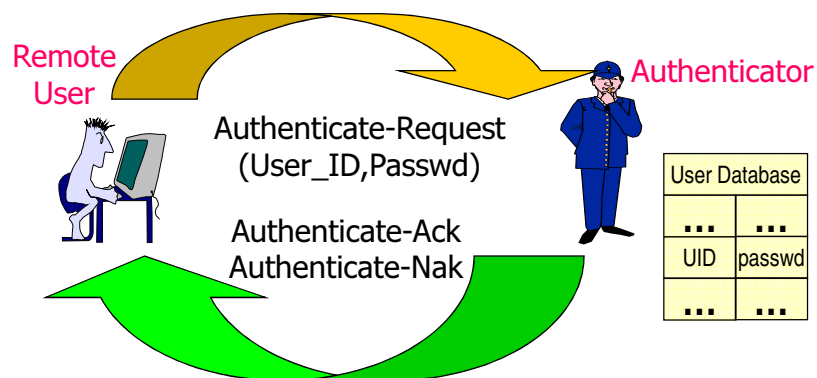
Giuseppe Bianchi

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



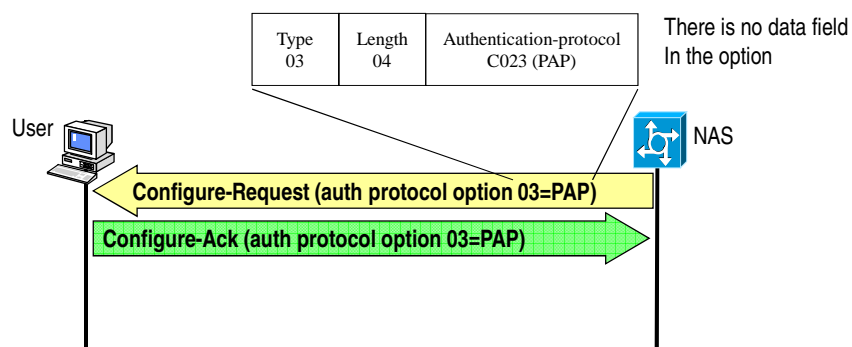
Giuseppe Bianchi

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.

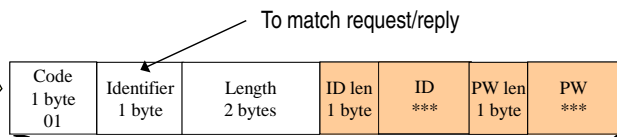
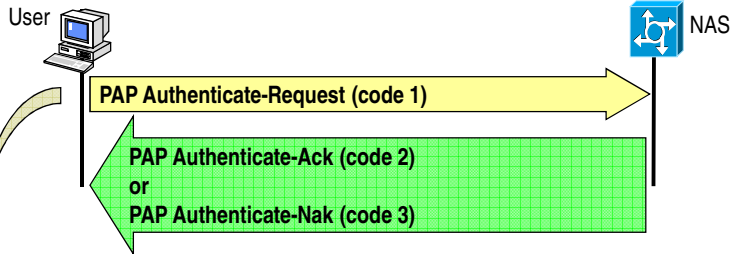
Giuseppe Bianchi

PAP Authentication Protocol option



Giuseppe Bianchi

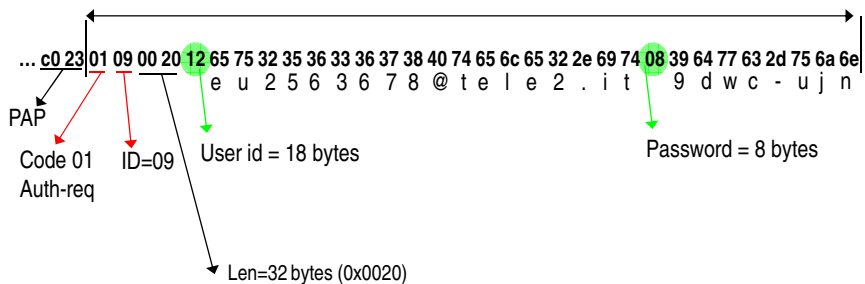
PAP packet format: auth-request



Flag	Address 11111111	Control 00000011	Protocol 0xC023 (PAP)	Information	FCS	Flag
------	---------------------	---------------------	--------------------------	-------------	-----	------

Giuseppe Bianchi

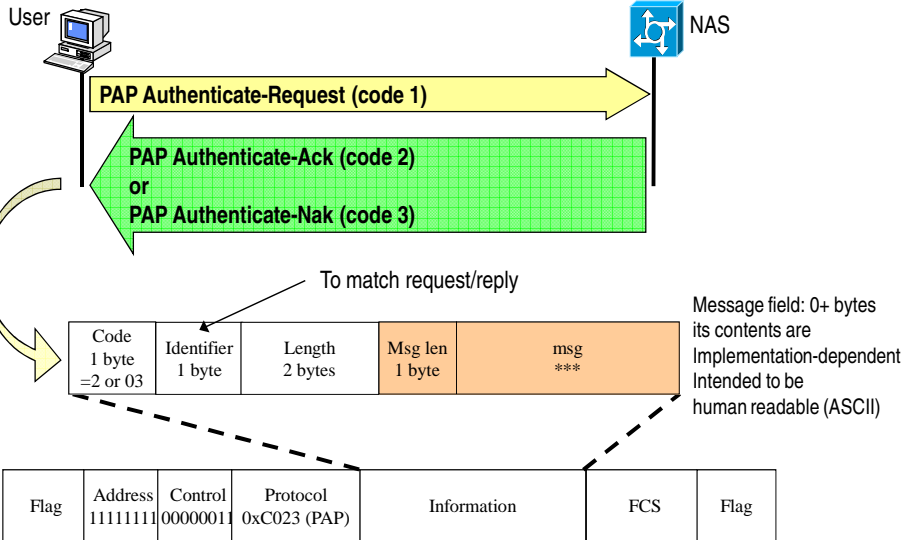
Auth-request example (real capture)



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

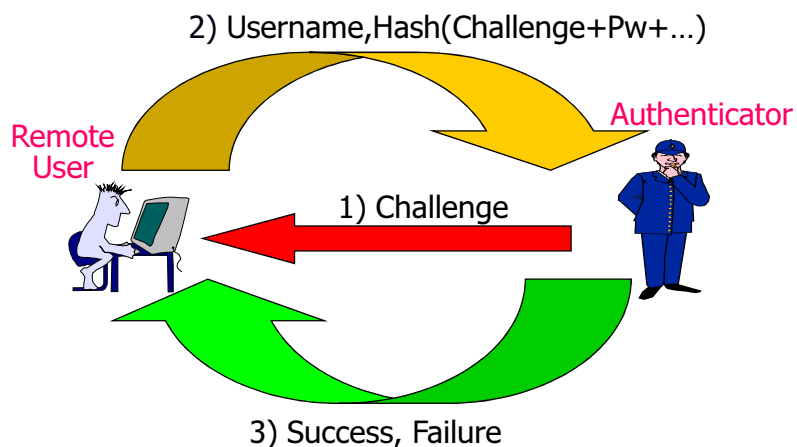
⇒ Much more safer than PAP

→ Conceptually identical to the approach currently adopted into actual cellular networks

⇒ GSM, UMTS

Giuseppe Bianchi

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!!

Giuseppe Bianchi

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

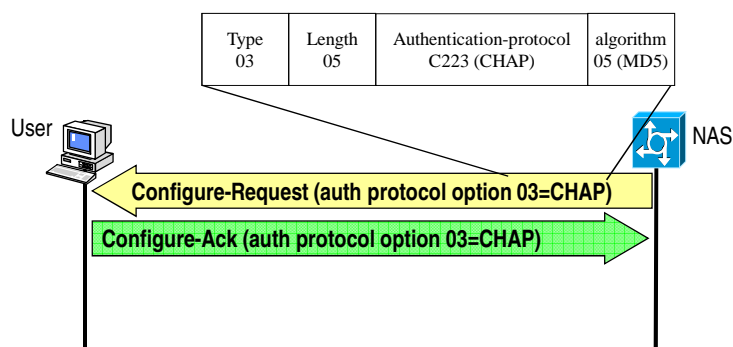
→ Cons:

- ⇒ 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

===== Giuseppe Bianchi =====

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)

===== Giuseppe Bianchi =====

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

```

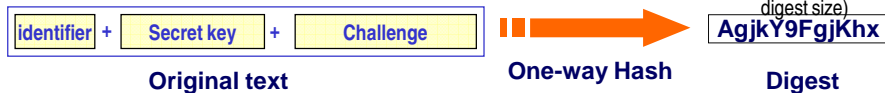
PPP Challenge Handshake Authentication Protocol (REAL TRACE EXAMPLE)
Code:      0x 01 (Challenge)
Identifier: 0x 01
Length:    0x 00 1f (31 bytes)
Value Size: 0x 10 (16 bytes)
Value:     0x 07 21 c9 b3 30 a6 f8 6f 52 ff 67 7f 07 3d 15 f5
Name:      MILZ-LNS-9 (10 bytes)
  
```

===== Giuseppe Bianchi =====

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)



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

```

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)
  
```

===== Giuseppe Bianchi =====

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

PPP Challenge Handshake Authentication Protocol (REAL TRACE EXAMPLE)

Code: 0x 03 (Challenge)

Identifier: 0x 01

Length: 0x 00 04 (4 bytes)

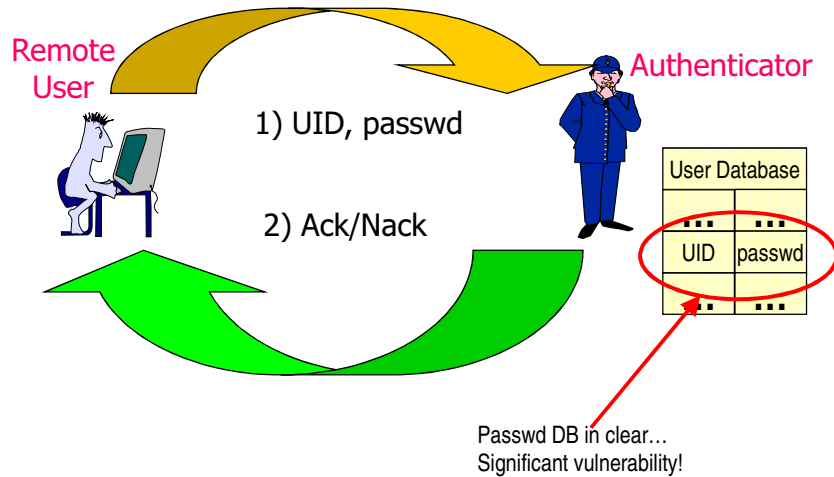
[no message in the considered implementation!]

==== Giuseppe Bianchi =====

Password based Authentication: Extensions

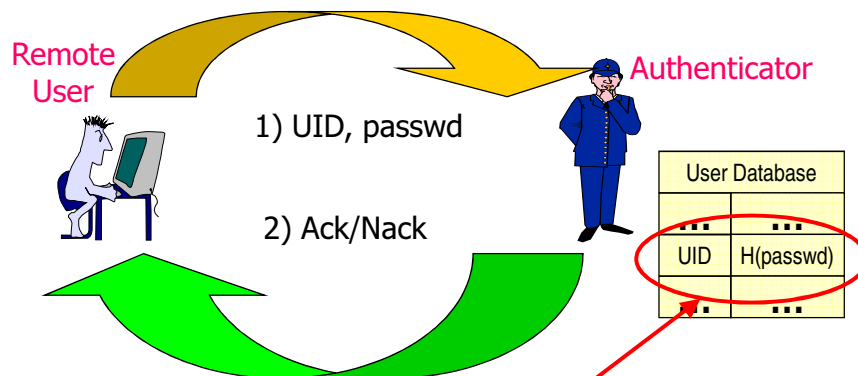
==== Giuseppe Bianchi =====

Passwd protection in DB?



Giuseppe Bianchi

Passwd protection in DB: storing passwd hashes!



Authenticator:

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

Giuseppe Bianchi

Dictionary attack...

→ Many users use predictable passwd

→ Dictionary attack:

⇒ Hashing does not help

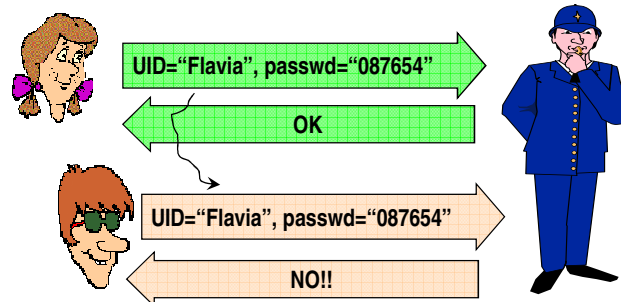
⇒ Will see in a dedicated laboratory!

==== Giuseppe Bianchi =====

One-time passwd

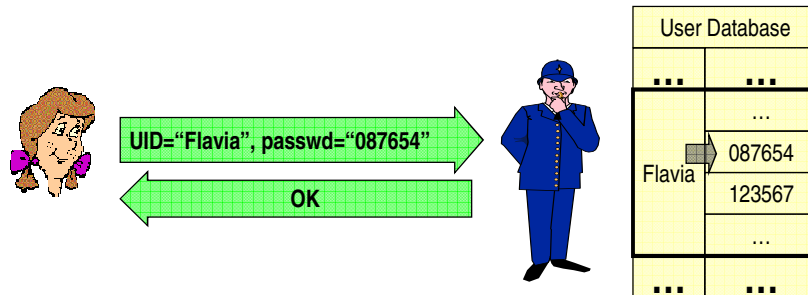
→ Is it possible to extend PAP so that user changes passwd at every (successful) attempt?

⇒ If it is, would prevent playback attacks



==== Giuseppe Bianchi =====

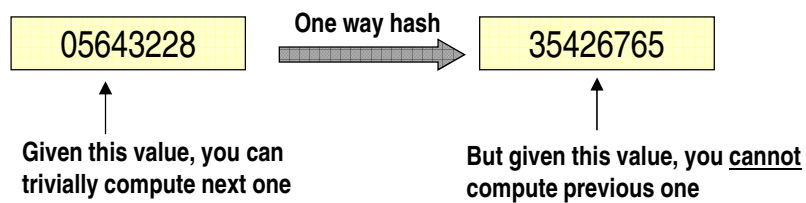
One-time passwd: trivial... but...



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

Giuseppe Bianchi

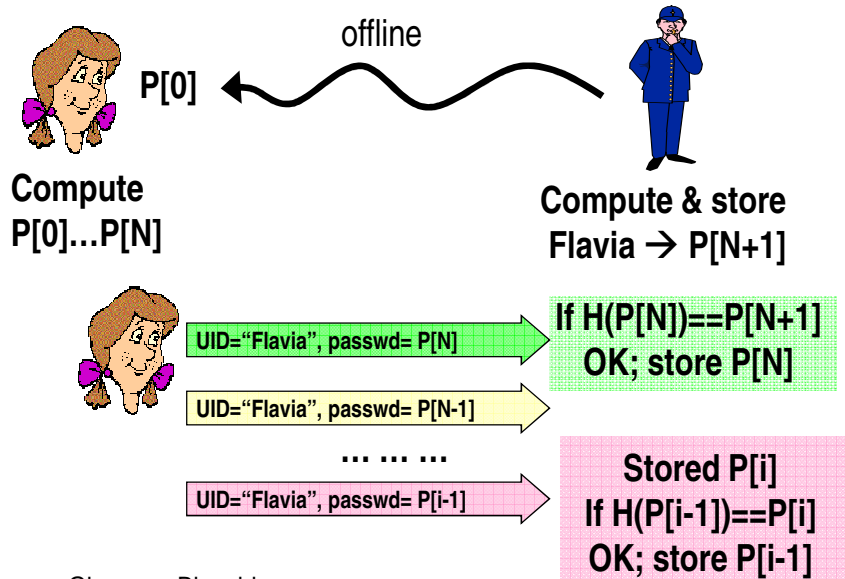
Idea: hash chains



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

Giuseppe Bianchi

One-time passwd: practical



One-time passwd benefits

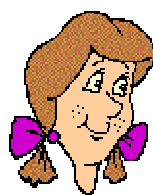
- **Passwd in clear = OK**
- **Authenticator only stores USED passwd**
 - ⇒ no way to predict next one (1-way hash)
- **Authenticator only stores 1 value**
 - ⇒ Same complexity as in ordinary PAP
- **Issues:**
 - ⇒ Large N to prevent frequent renegotiation
 - ⇒ Client size = vulnerable (must store passwd seed or whole vector)

Giuseppe Bianchi

Issues with CHAP

Giuseppe Bianchi

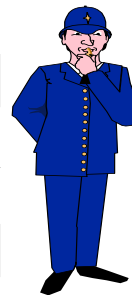
Basic CHAP vulnerability



CHALLENGE = 135623

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

ACK or NACK



→ **Authenticator MUST store passwd in clear!**

⇒ Otherways no way to compute $H(\text{id}, \text{pw}, \text{challenge})$

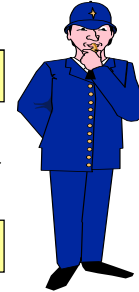
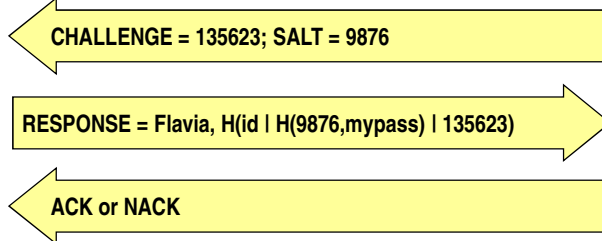
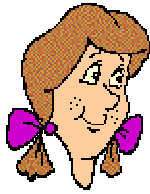
→ **Authenticator storage = straightforward target for attack!**

⇒ Even worse than PAP!!

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

Giuseppe Bianchi

Idea: "salt"



→ Attacker may only access to "salted" passwd

⇒ Different salt for different authenticator servers

→ breaking one != breaking all

⇒ Refresh DB periodically

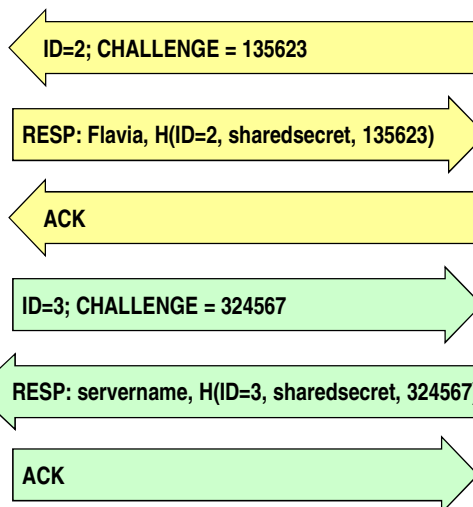
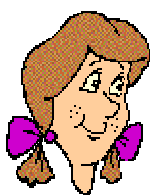
→ Someone must take care of this... more later

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

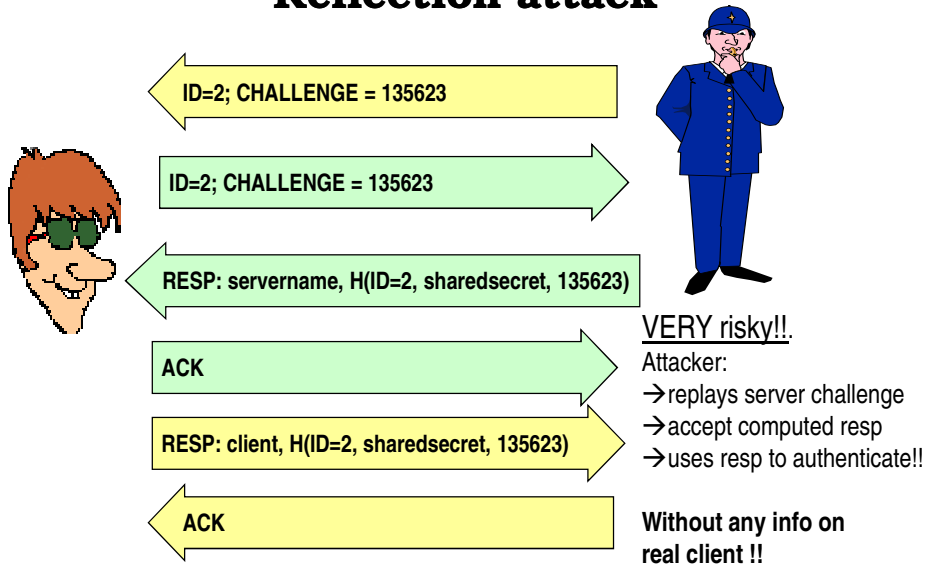


Usage of a shared Secret... good idea, Easy to manage!

Good idea??

Giuseppe Bianchi

CHAP and mutual authentication /2 Reflection attack



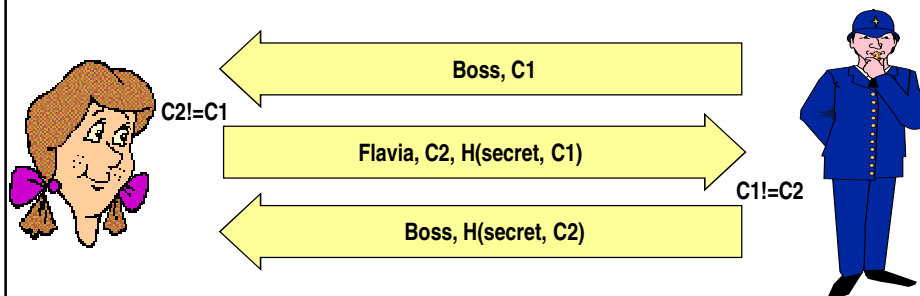
Giuseppe Bianchi

Mutual authentication with Challenge-Response

(just some hints... no full analysis)

Giuseppe Bianchi

Basic idea



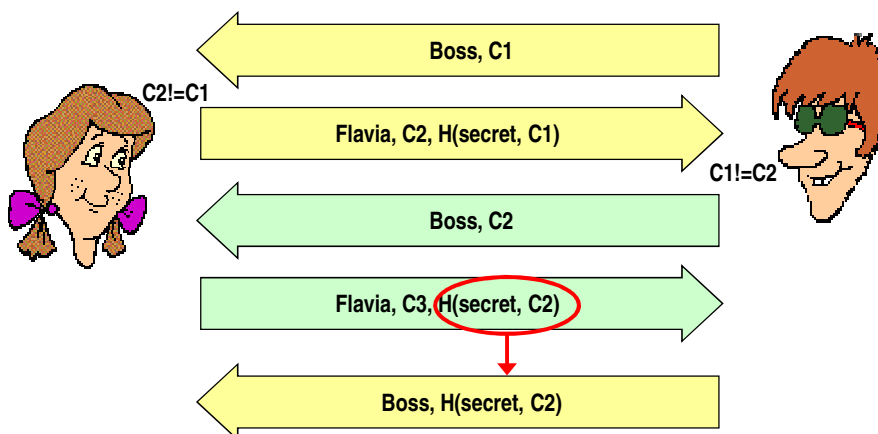
Flavia shows knowledge of secret over C_1

Boss shows knowledge of secret over C_2

Giuseppe Bianchi

Reflection!

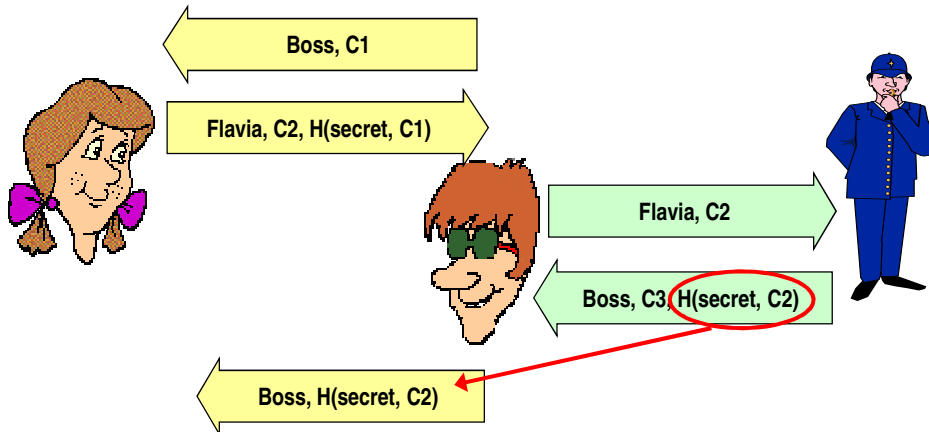
→ Does not work



Giuseppe Bianchi

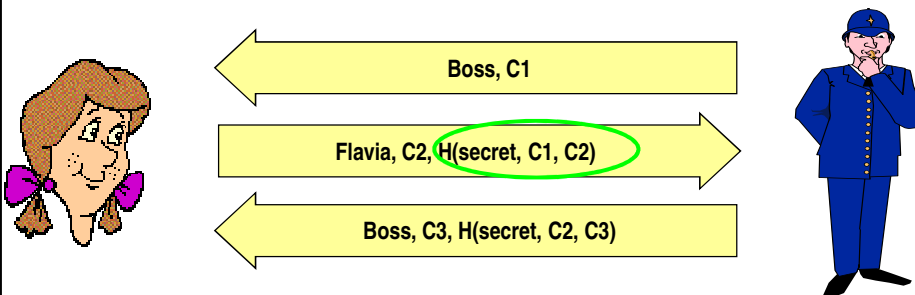
What if reflection is prevented by protocol status?

→ Attacker may use “other” party!



Giuseppe Bianchi

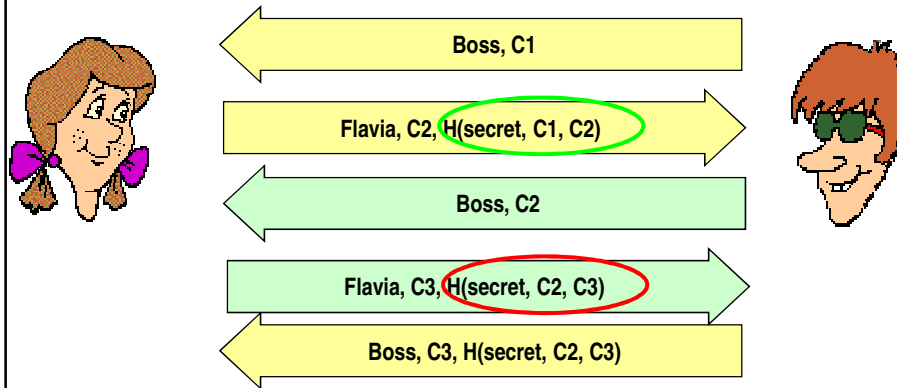
Let's try to fix this



Chaining challenges! Add dependency between challenges in same handshake

Giuseppe Bianchi

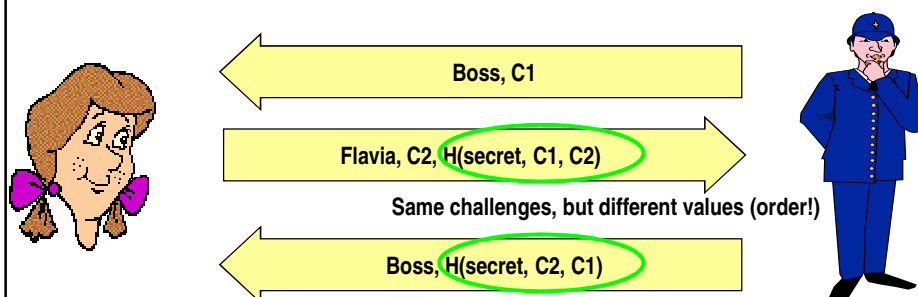
Does not work



Too many nonces!!

Giuseppe Bianchi

Minimize nonces



Same challenges, but different values (order!)

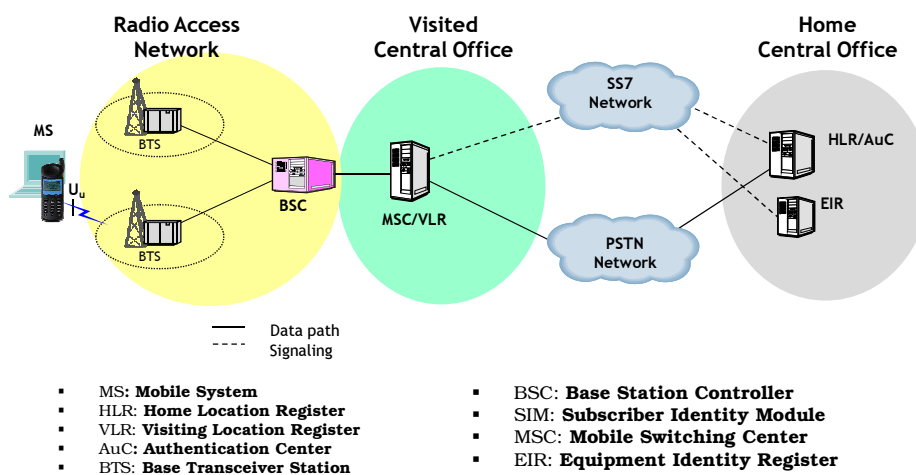
No reflection possible anymore

Giuseppe Bianchi

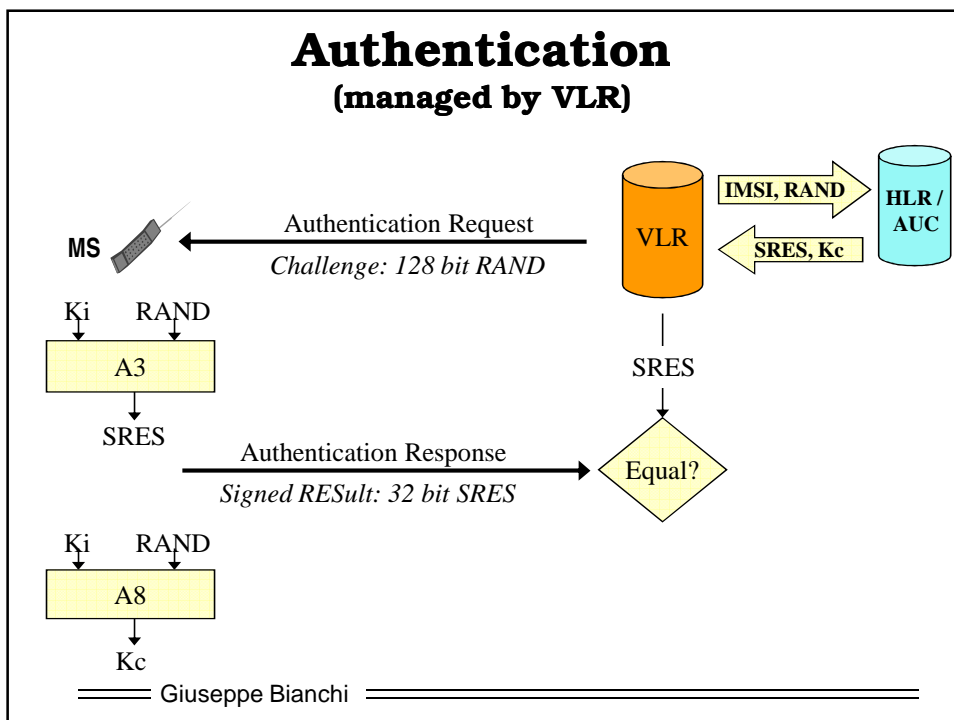
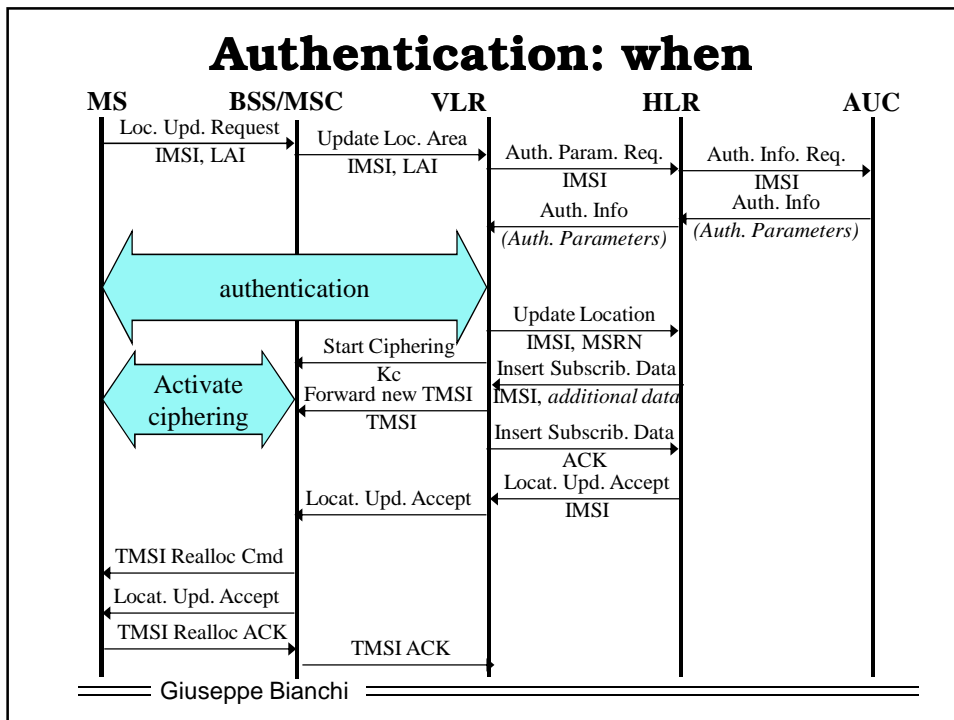
Challenge-Response in GSM authentication

Giuseppe Bianchi

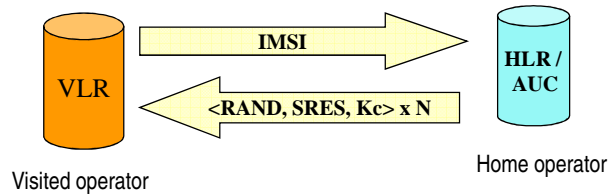
GSM essential components



Giuseppe Bianchi



Triplets (Authentication Vector)



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

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

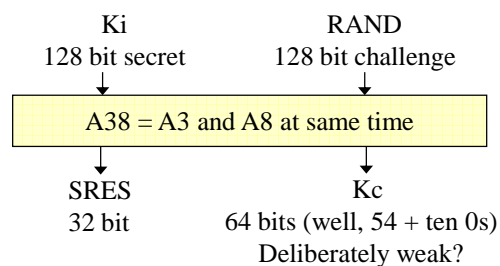
→ **IMPORTANT: VLR does NOT need to know authentication algo used (A3, A8)**

⇒ Triplet contains computed result by AuC

⇒ A3, A8 run inside the SIM (given by operator)

===== Giuseppe Bianchi =====

Authentication: details



→ **Challenge response with:**

⇒ Challenge

→ RAND

⇒ Secret

→ Ki

⇒ Hash

→ A3 algorithm

===== Giuseppe Bianchi =====

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

===== Giuseppe Bianchi =====

Over the air attack

→ No mutual authentication!

→ Rogue BTS may easily perform the attach

- ⇒ run it for sufficient time

===== Giuseppe Bianchi =====

UMTS authentication: AKA Authentication and Key Agreement

(Simplified for our purposes)

==== Giuseppe Bianchi =====

Major differences with GSM

→ Mutual authentication!

⇒ Optimized...

⇒ Guaranteed freshness for auth parameters

→ More comprehensive security

⇒ More keys and several extra details

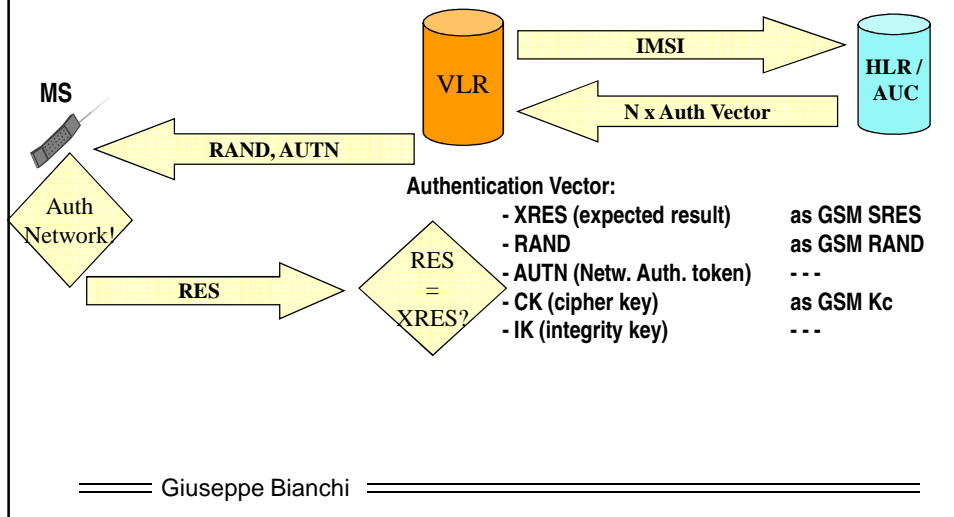
→ (we will not focus on this)

→ Algorithms FIRST scrutinized by research community, THEN selected

⇒ The opposite of security by obscurity ☺

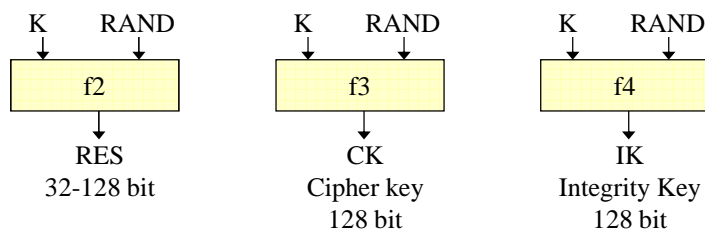
==== Giuseppe Bianchi =====

Authentication Vector



MS authentication

→ Usual challenge response, but different (public) algorithm



Network Authentication

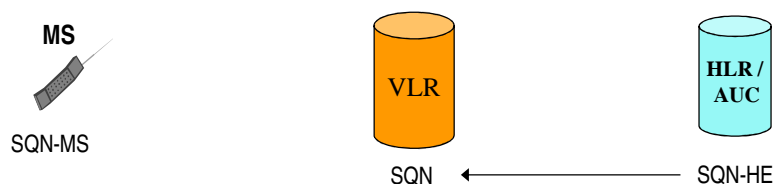
→ MS should send a nonce...

⇒ 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



Giuseppe Bianchi

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

Giuseppe Bianchi

Network authentication: AUTN format

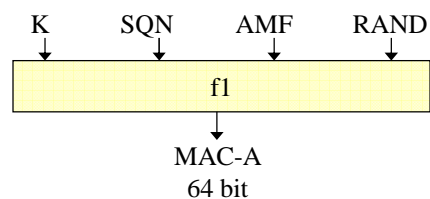
SQN	AMF	MAC-A
Sequence number	Auth & key mgmt field	Message Auth code

48 bit

16 bit: carries info on
which algo or key to
use if choice available
(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**



===== Giuseppe Bianchi =====

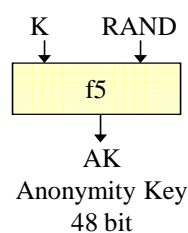
Minor detail: protecting SQN!

→ A privacy problem:

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

→ Solution: mask SQN with Anonymity Key

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



===== Giuseppe Bianchi =====