Lecture 3.1:

Handling Remote Access: RADIUS

Remote Authentication Dial In User Service

ancora oggi usato ! call -> modem -> internet

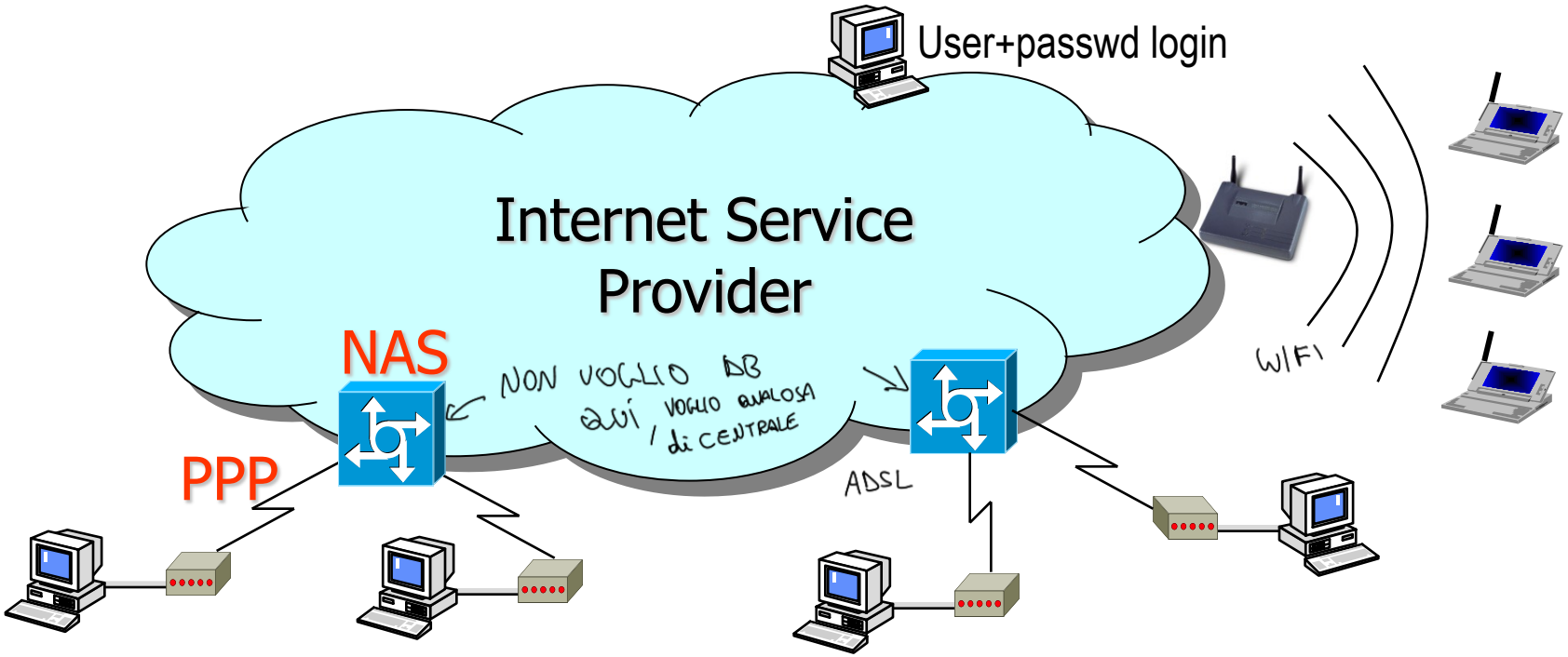
Recommended reading:
RADIUS is standardized in RFC 2865, June 2000
Radius (poor!) security analyzed in Joshua Hill, "An analysis of the RADIUS Authentication Protocol"

Motivation

- **Managing large-scale networks: a nightmare**
  - Multiple NAS, multiple access services
  - Not only authentication, but also service-specific configuration assignment

(centralizzo DB)

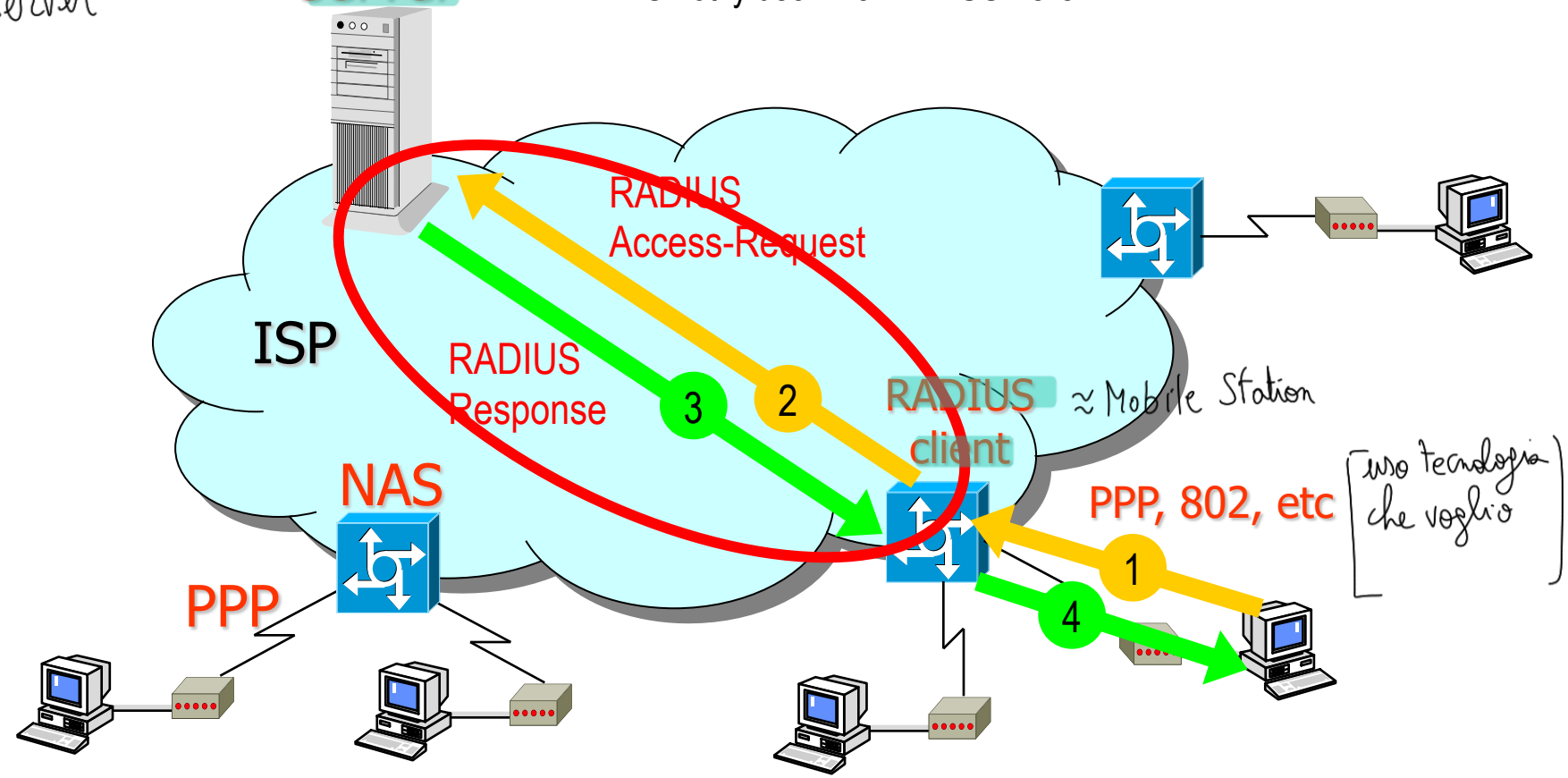
- **Best achieved by managing single user "database"**



RADIUS: where?

1. User sends authentication attributes to NAS: **no RADIUS here**
2. NAS wraps them into Access-Request → to RADIUS Server
3. RADIUS Server response: OK, NO, Challenge (for some AUTH)
4. NAS notify user: **no RADIUS here**

authentication server ≈ RADIUS server



PPP: Point to Point Protocol

RADIUS: AAA protocol

è un "container"

➔Provides centralized AAA functionalities

- ⇨Authentication
  - ➔are you really the one you claim to be?
- ⇨Authorization
  - ➔Do you have permissions to access a service?
- ⇨Accounting
  - ➔what are you currently doing/using/paying?
    - » Transmitted bytes, billing, etc

RADIUS: client-server protocol

➔ **Radius client: the NAS!!**

⇨ Don't confuse it with the end user!!

➔ **Client-Server protocol**

⇨ Based on UDP/IP

⇨ Server port 1812 (client port ephemeral, as usual in C/S)

➔ **(Logically) centralized**

⇨ 1 primary server (0+ secondary servers - replicated)

NAS non lo vede

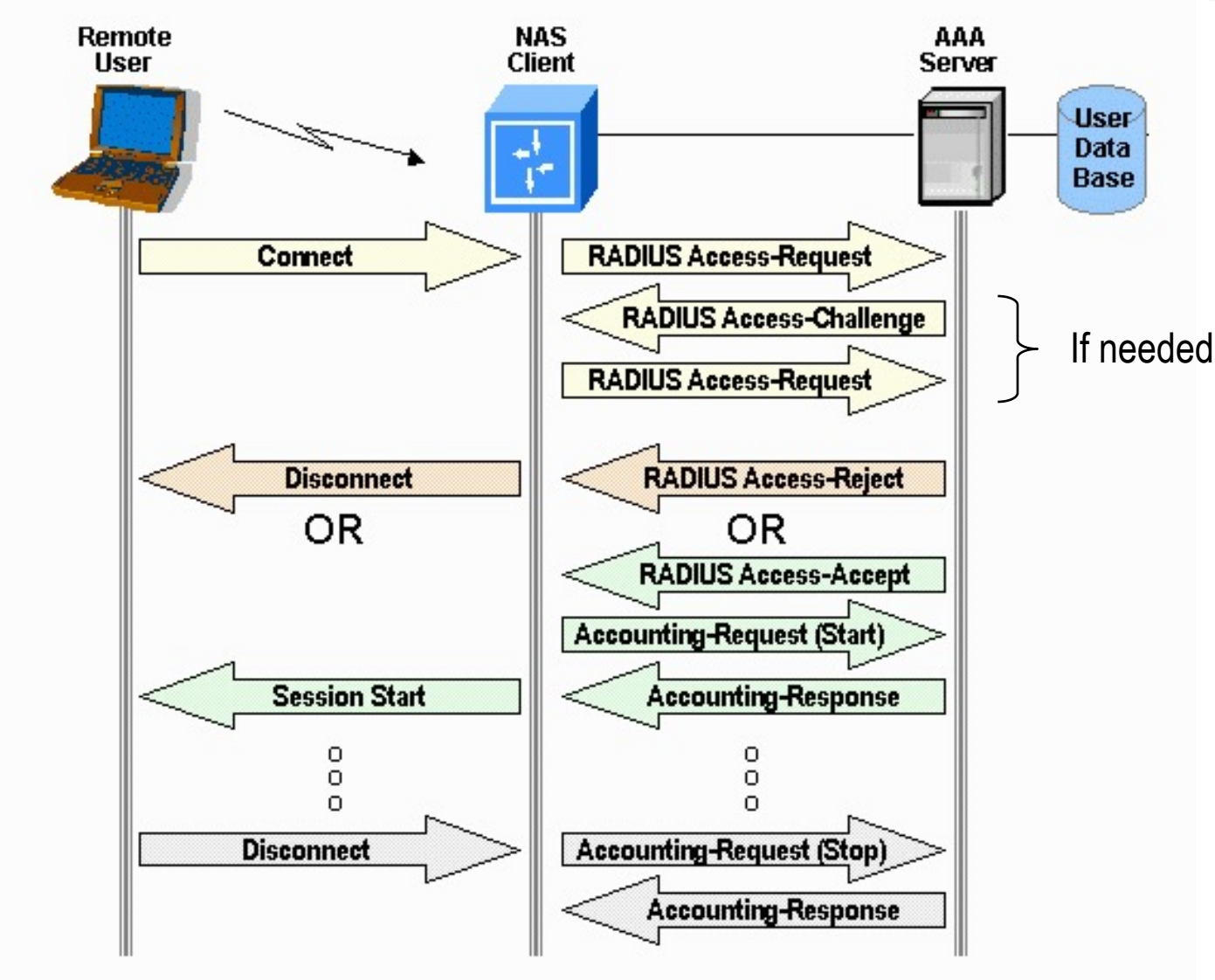
➔ Management of replicated servers ➔ implementation dependent

⇨ Server may in turns act as a proxy

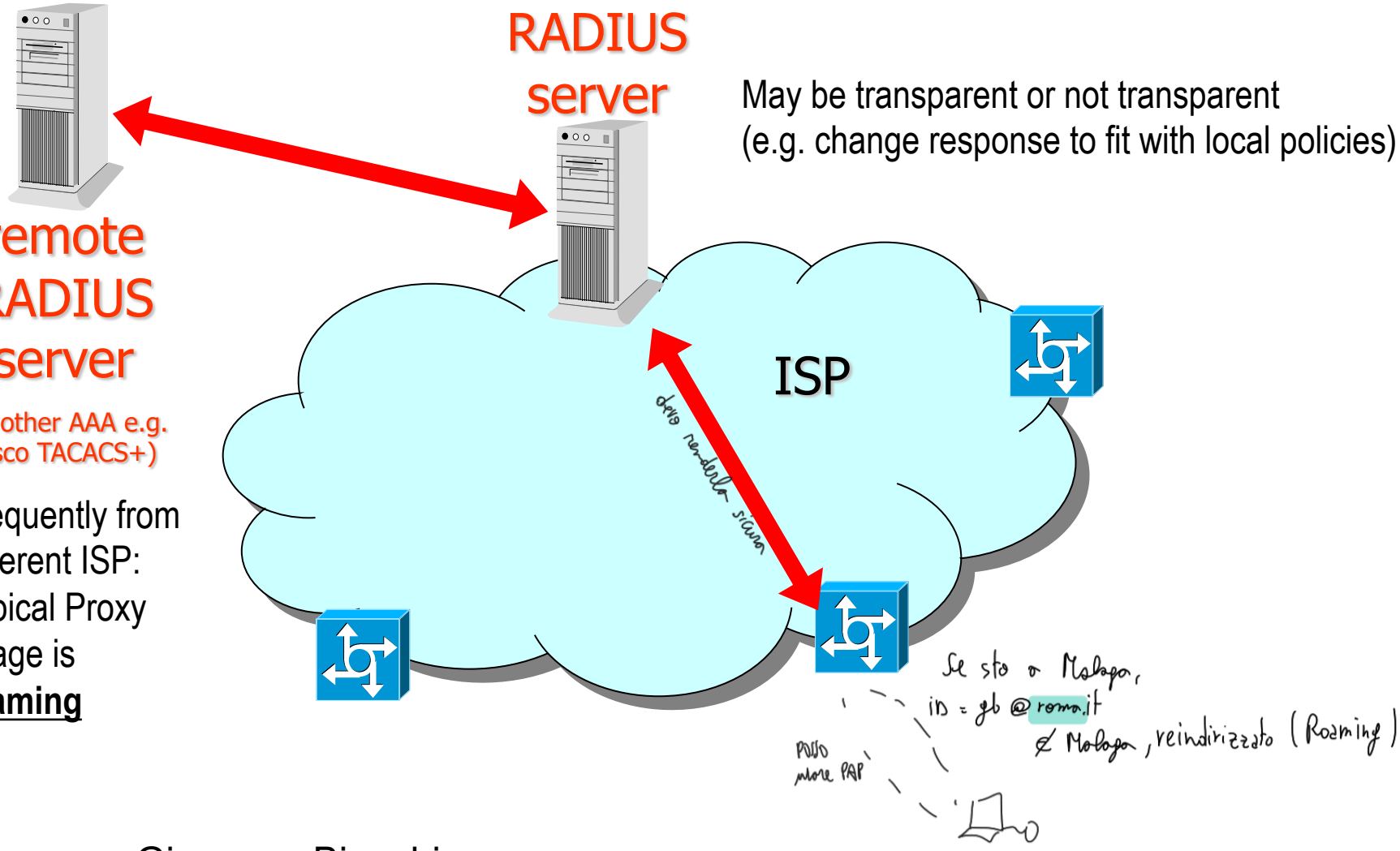
Se ∃ ransomware nel Radius Server : RIP

SKIP

Message exchange (example)



Proxy Operation



RADIUS architecture

- **RADIUS Server application**

- **Registered User Database**
  - For each entry (user_name), contains (at least):
    - Authentication information (secrets)
    - Authentication Method
    - Authorization attributes (access profile per each user)

- **Client database** (cosa possono fare)
  - Clients which are entitled to communicate with the server
  - BE CAREFUL: DO NOT confuse (radius) clients with (end) users

- **Accounting Database**
  - Whhen radius used for accounting
    - Frequently used only for authentication

RADIUS Security features

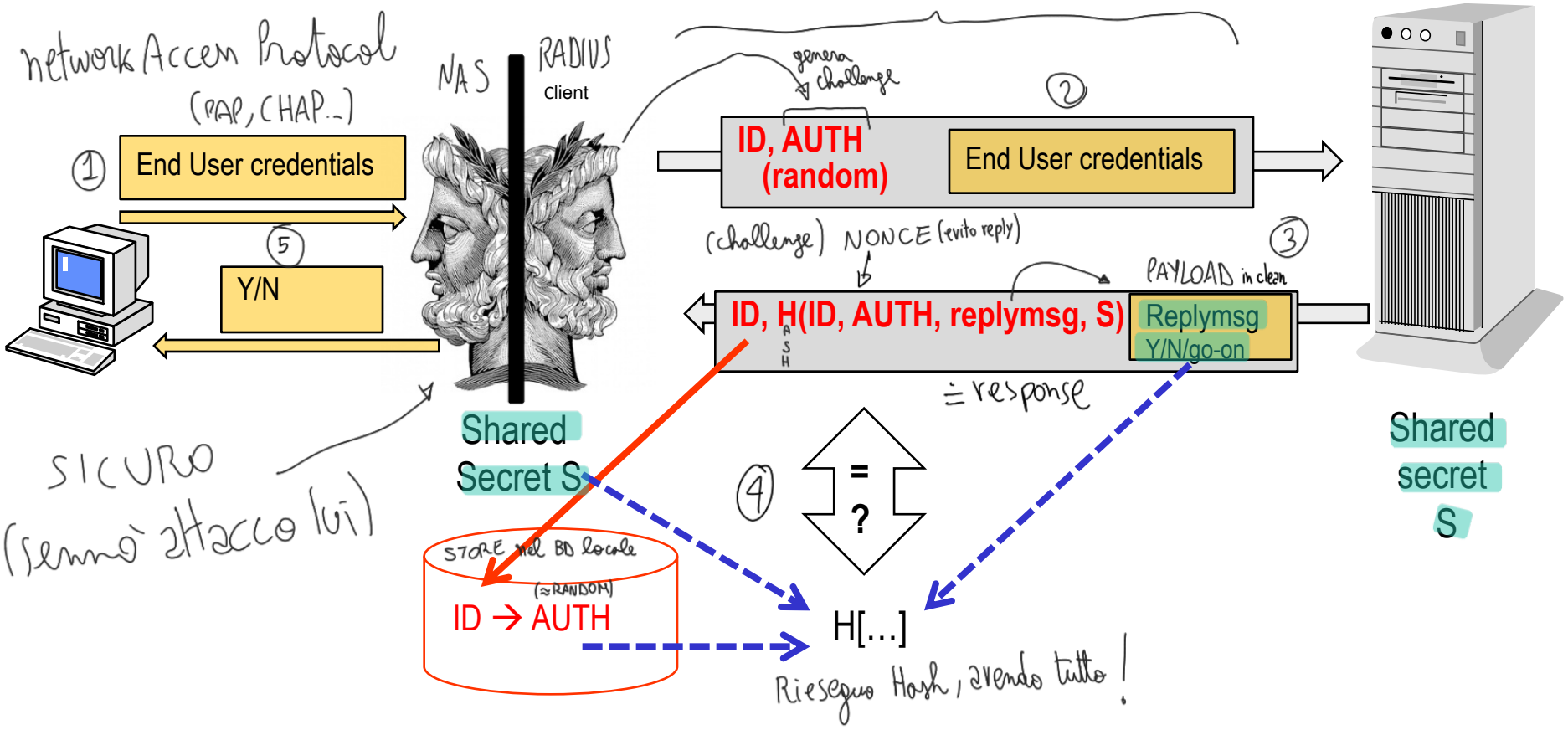
- **Per-packet authenticated reply** (per risolvere spoof attack)
  - Shared-secret based
  - No transmission of secrets (CHAP-like, more later)
  - but…
    - Only reply is authenticated…
    - Hash-based, not HMAC-based…
    - Very specific hash function, MD5
    - Often low-entropy shared key..

p. 3:
- a volte pw non in chiaro (CHAP), a volte sì (PAP).
- Radius client no dB e chiede al server, posso intercettare richiesta e auto-rispondere (non protetta)

La reply deve essere autenticata

- **Encrypted user password transmission**
  - Custom mechanism, MAI FARLO!
  - Same shared key used for authentication!!

RADIUS authenticated reply: concept



Sort of challenge-response:

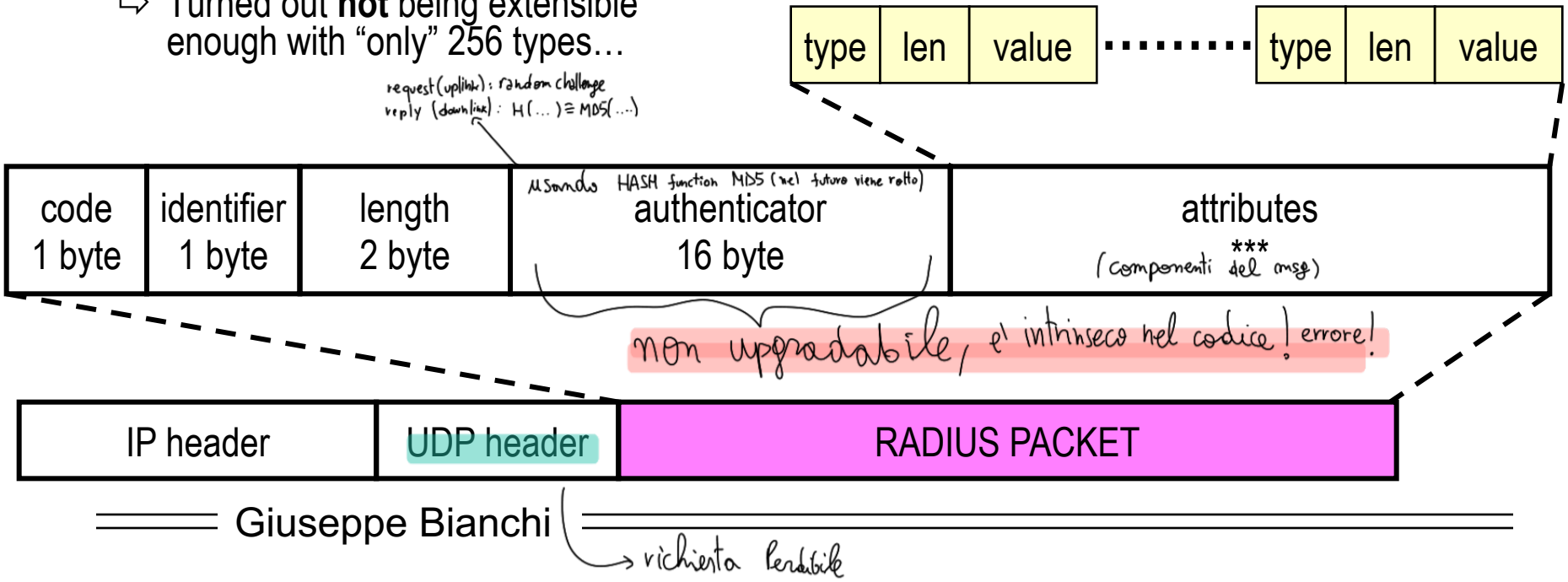
- challenge is the request authenticator (AUTH)
- response also includes (hence validates) reply message

packet format

- **Code: type of radius packet**
- **Identifier: match requests with responses**
  - IP src and UDP src also help matching (se perdo pkt)
- **Length**
  - minimum 20, maximum 4096
- **Authenticator:**
  - used to authenticate reply from server
  - Used also as nonce for password encryption
- **Attributes: extensible information field**
  - Turned out **not** being extensible enough with "only" 256 types…

Code (dec)	Packet
1	Access-Request
2	Access-Accept
3	Access-Reject
4	Accounting-Request
5	Accounting-Response
11	Access-Challenge ≐ LET'S GO ON [3° tipo di risposta]

nome fuorviante



richiesta perdibile

Authentication field (16 byte)

- **In Access-Request (C→S)**
  - 16 randomly generated bytes
    - unpredictable and unique (over the lifetime of shared C/S secret)
      - » To avoid replay attack
- **In response packets (S→C Accept/Reject/Challenge)**
  - One-way **MD5 (!!)** hash of (hardcoded)
    - the request ID
    - the request authenticator,
    - the shared secret,
    - the packet response information
      - » **Response** packet is signed! Otherwise spoofed server's reply possible!
  - Specifically:
    MD5(Code | ID | Length | RequestAuth | Attributes | Secret)

response — presa da request — response

shared

tutto il pkt "reply" ≡ response

Attributes (at a glance)

- **Information and configuration details**
  - in request and/or reply (accept/reject/challenge) packets
- **Any number of attributes in a packet**
  - Length field → end of attributes payload
- **Order of attributes does NOT matter**
- **Some attributes may appear more than once**
  - effect is attribute-specific (here order may matter!)
- **Up to 2^8 attributes (1 byte type field):**
  - Type 0: reserved
  - Type 1-191: IANA (public) assigned/assignable
  - Type 192-240: for private use
    - Type 192-223: experimental
    - Type 224-240: implementation-specific
  - Type 241-255: reserved → finiti questi, RIP
- **Extensible Protocol**
  - New attribute values can be added without disturbing existing implementations

1	User-Name
2	User-Password
3	CHAP-Password
4	NAS-IP-Address
5	NAS-Port
6	Service-Type
7	Framed-Protocol
8	Framed-IP-Address
9	Framed-IP-Netmask
10	Framed-Routing
11	Filter-Id
12	Framed-MTU
13	Framed-Compression
14-16	(for Login service)
17	(unassigned)
18	Reply-Message
19	Callback-Number
20	Callback-Id
21	(unassigned)
22	Framed-Route
23	Framed-IPX-Network
24	State
25	Class
26	Vendor-Specific
27	Session-Timeout
28	Idle-Timeout
29	Termination-Action
30	Called-Station-Id
31	Calling-Station-Id
32	NAS-Identifier
33	Proxy-State
34-36	(for LAT)
37-39	(for AppleTalk)
40-59	(res. for accounting)
60	CHAP-Challenge
61	NAS-Port-Type
62	Port-Limit
63	Login-LAT-Port

SKIP

Access-Request

- **Typically contains:**
  - Who is the user
    - User-Name
      - Mandatory: search key to access the user database
  - Password
    - User-Password
    - CHAP-password (when CHAP employed)
  - An identifier of the RADIUS client
    - NAS-IP or NAS-identifier
      - user might access only a subset of NAS
  - An identifier of the port the user is accessing
    - NAS-Port (if the NAS has ports)
      - Wi-Fi: Logical association
      - Dial Up: physical (modem) port# receiving the user call
    - User might be restricted to access only specific ports

Password encryption

solo se uso PAP, con attributo '2' user-password

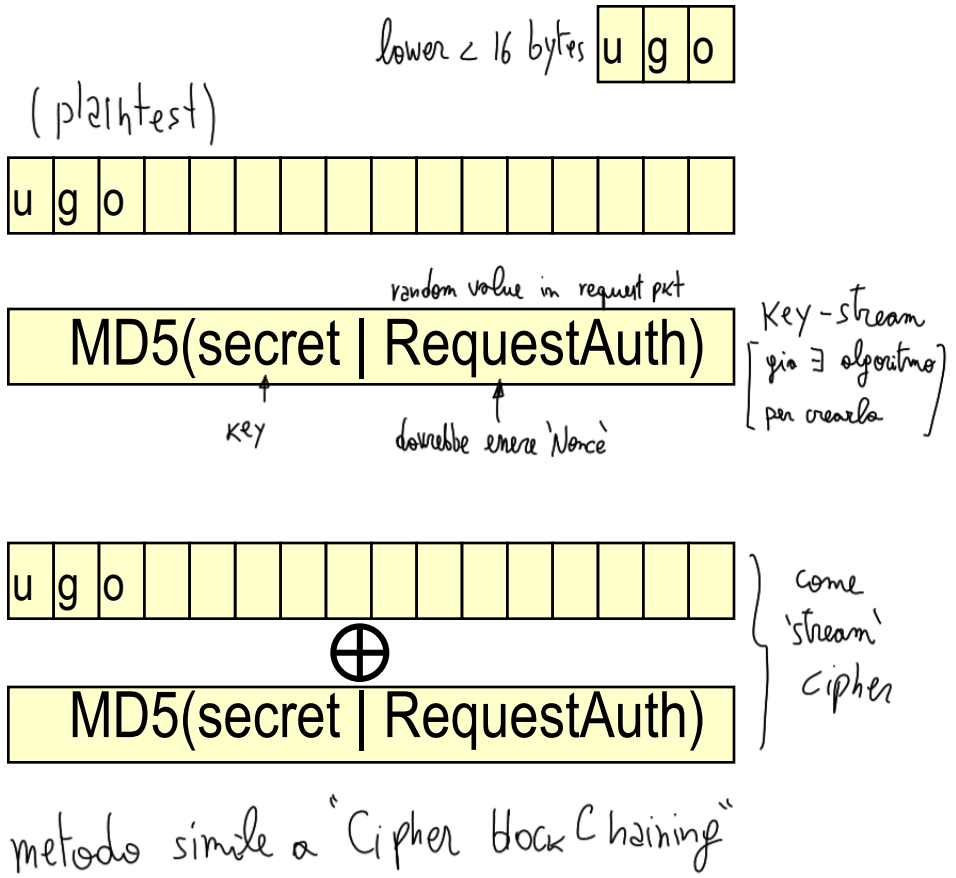
creato, mai buona idea.

Native User-Password

Step 1: padding to 16 bytes

Step 2: generate a 16 bytes hash using key and the content of the authenticator field of the request

Step 3: XOR padded passwd & hash



metodo simile a "Cipher Block Chaining"

If passwd longer than 16 characters:

Step 4: compute MD5(secret | result of previous XOR) and

Step 5: XOR with next segment of the passwd

hanno riusato stesso keystream!

SKIP

Access-Accept

- **Positive server response**
  - ⇨ User authentication credentials OK
- **Contains all the service-specific configuration**
  - ⇨ Including the Service-Type attribute
  - ⇨ Complemented with other service-related configuration parameters
    - → E.g. IP address, mask, etc

Login	The user should be connected to a host.
Framed	A Framed Protocol should be started for the User, such as PPP or SLIP.
Callback Login	The user should be disconnected and called back, then connected to a host.
Callback Framed	The user should be disconnected and called back, then a Framed Protocol should be started for the User, such as PPP or SLIP.
Outbound	The user should be granted access to outgoing devices.
Administrative	The user should be granted access to the administrative interface to the NAS from which privileged commands can be executed.
NAS Prompt	The user should be provided a command prompt on the NAS from which non-privileged commands can be executed.
Authenticate Only	Only Authentication is requested, and no authorization information needs to be returned in the Access-Accept (typically used by proxy servers rather than the NAS itself).

Possible values of the Service-Type option

SKIP

Access-Reject

➔**Two main reasons:**

⇨Authentication failed

⇨1+ attributes in the request were not considered acceptable (authorization failed)

Access-Challenge

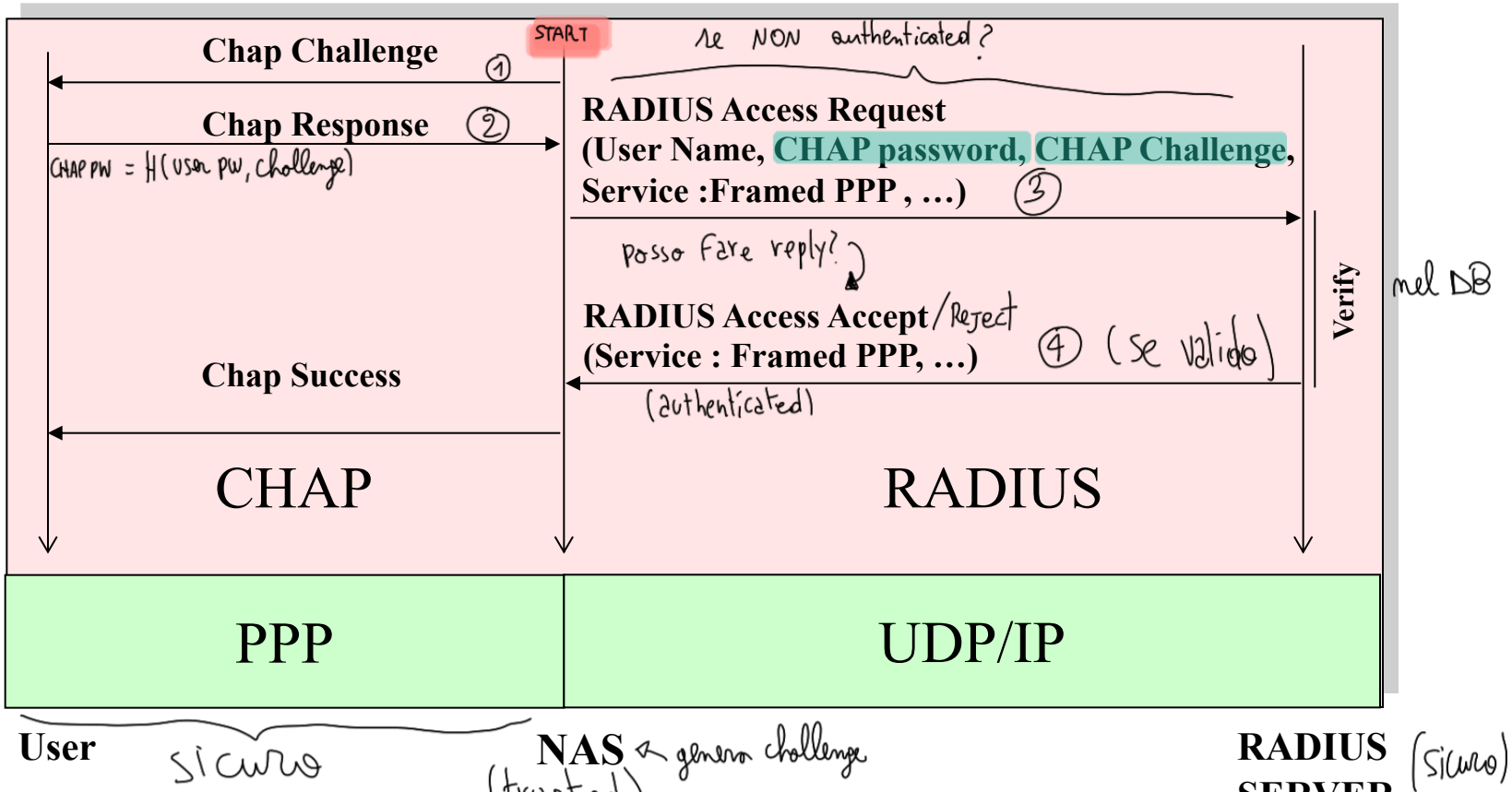
(= let's go on)

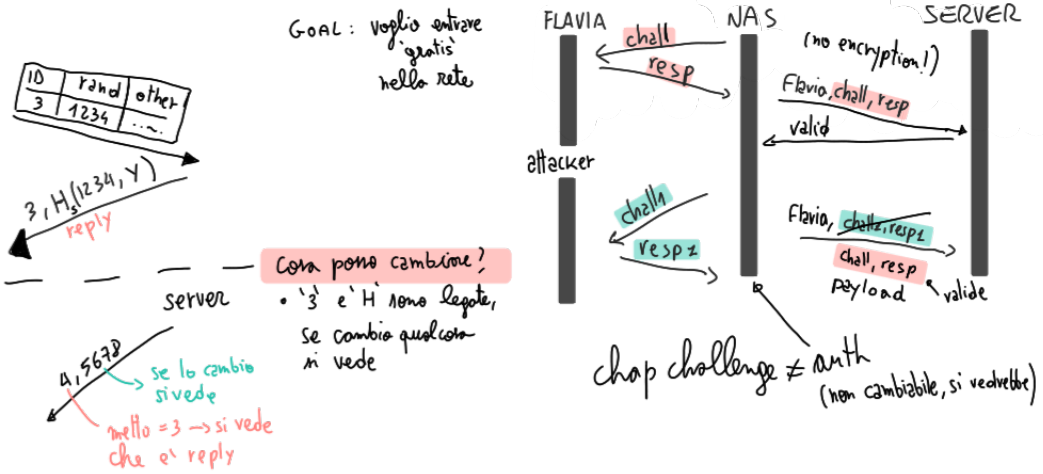
(response msg : "andiamo avanti")

- **Used whenever the server wants/needs the user to send a further response**
  - E.g. a challenge/response authentication mechanisms
    - →Not necessarily CHAP (see CHAP support later on)! Could be RADIUS support for GSM/UMTS authentication!
  - E.g. prompting the user to enter a password
- **Challenge typically contains**
  - One or more reply-message attributes
    - » Which MAY be used in a very flexible manner
    - →May contain text to be prompted to the user
    - →May contain an explicit authentication challenge
- **NAS collects response from the user and sends a NEW Access-Request**
  - New ID
  - New User-Password - contains the user response (crypted)
- **Based on this, server accepts or rejects or send another challenge**

PPP CHAP support with RADIUS

- **CHAP challenge locally generated by NAS**
  - No need to know user password for this!
- **CHAP challenge + response sent to RADIUS server**
  - RADIUS server retrieves user password from database, computes and compares CHAP response

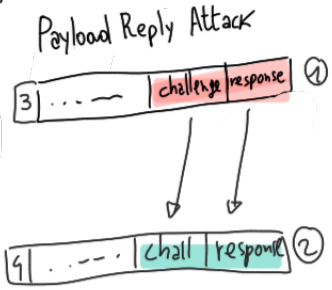




COME FARE:

1) osservo e catturo overhead |challenge, response| overhead

2) quando Flavio finisce, entro io e finge di essere Flavio, arrivo challenge ed eseguo la challenge, che viene incapsulata in nuovo pacchetto: sostituisco con vecchia challenge! Response è "fresh", anche se reply payload.



1ª soluzione: (PATCH)

- NAS mantiene { ID → auth
  End user credentials P
- Server risponde ma fa Hash includendo anche P H(ID, auth, reply, resp, S) così è legata alla richiesta NAS → server avente: ID/auth/end user credentials

2ª soluzione:

autentifico replay/response, infatti REQUEST non è autenticata, non lego richiesta e risposta.

Vulnerable to message sniffing and modification

- **Clear-text protocol → privacy issues**
  - ⇨ User-Name, Calling-Station-ID, NAS identification, location attributes sent in the clear

- **Access-Request not authenticated**
  - ⇨ Access requests may be forged/modified (MITM)
  - ⇨ Solution:
    - → Message-Authenticator attribute
    - → Mandatorily used with EAP (only, though)

Ricordo che 16 byte 'authenticator' non toccabile, allora lo estendo nel campo 'attributes', in cui metto il 'TAG', che contiene authentication of request msg. Rimane "opzionale"

Extension:

Message-Authenticator

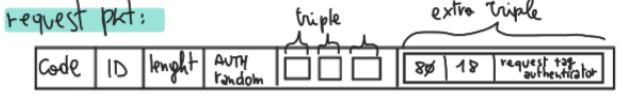
➔ Message signature

- ⇨ Primarily for Access-Request messages
  - ➔ Since they are not authenticated
  - ➔ Of course can be used also in Reject/Accept/Challenge packets

(extensible Auth. Protocol) = contenitore per auth. protocol (è una "sintassi")

- ⇨ MUST be used when EAP used with RADIUS (RFC 2869)
  - ➔ May (of course) be used with other authentication methods

➔ Usual attribute syntax

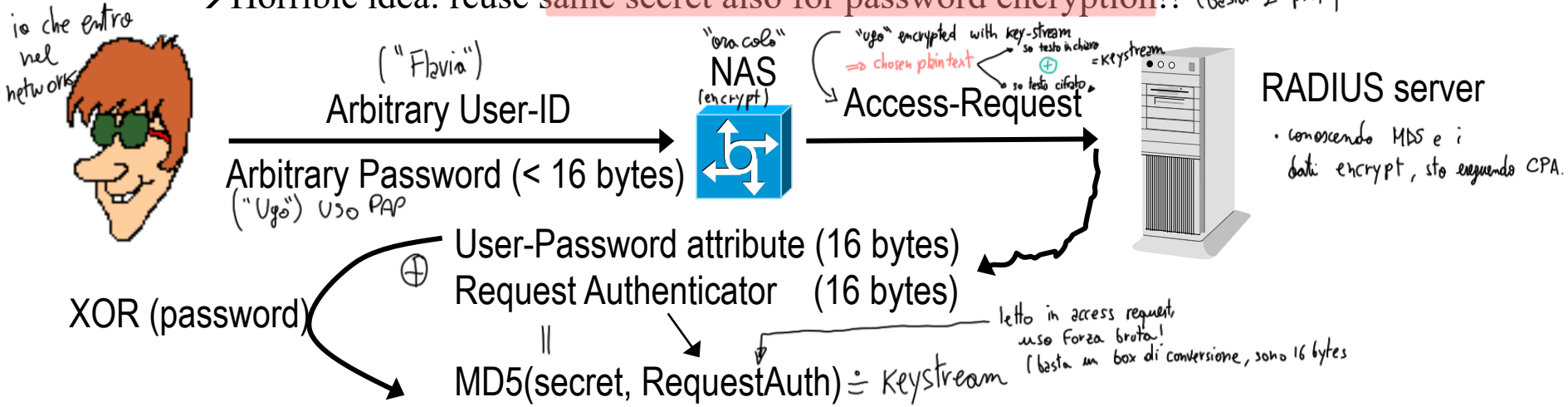


Type=80	Len=18	Authenticator (16 bytes)

- ⇨ Authenticator = HMAC-MD5(request packet) = HMAC-MD5(type | ID | len | RequestAuth | attributes)
- ⇨ In computation, Authenticator = 0000.0000.0000.0000 (serve forse hash di me stesso!)
- ⇨ Shared secret used as key for the HMAC-MD5 hash

Dictionary attack to shared secret

- **Usually low-entropy shared secret**
  - Many implementations only allow ASCII shared-secrets
- **And often a single one is used for the entire network!!**
  - Fun story: Fonera hack! http://stefans.datenbruch.de/lafonera/
  - Lesson: use one shared secret per each client (look up using client IP address)
- **Many hooks for offline dictionary attack:**
  - Intercept any request-response pair
    - Request gives you random ReqAuth
    - RespAuth = MD5(Code | ID | Length | ReqAuth | Attributes | Secret)
      - » Secret placed at the end: MD5 state pre-computation makes attack easier!
  - Not even need to get a pair, just a request with user passwd suffices!
    - Horrible idea: reuse same secret also for password encryption!!



scoprire pw user

Attacking the password of a user

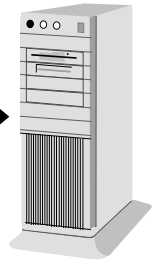
FIRST STEP: as previous case, but with valid user ID:

Victim User-ID

Arbitrary Password (< 16 bytes)

NAS

Access-Request



RADIUS server

encrypt arbitrary pw

User-Password attribute (16 bytes)

XOR (password)

MD5(secret, RequestAuth) Valid keystream

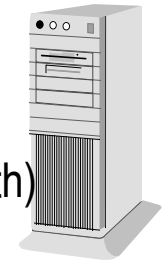
posso creare pkt contenente:
- RequestAuth
- Valid Encrypt Pw

questo perché non ho limiti sui tentativi né sul riuso di RequestAuth (brute force)

SECOND STEP: Attacker now able to "encrypt" the user password!! May exploit:
1) lack of upper limit on authentication rate at server-side (limits imposed on clients are by-passed)
2) RADIUS servers typically do not check for authenticator reuse
Works only with 16 or less byte passwords (most cases)

Spoofed Access-Request, with:

New-passwords XOR MD5(secret, RequestAuth)



poor PRNG implementations

- **Security of radius:**
  - Requires **uniqueness** of the Request Authenticator
    - Must be a nonce!

- **Some implementations:**
  - poor Pseudo-Random Number Generators (PRNGs)
    - Often non-crypto generators: horrible!!
    - Short cycles, predictability, …

- **THREE checks (when facing a weak PRNG)**
  - what is the «cycle»?
  - What about predictability?
    - E.g. LCG, if you know one value you know all!!
  - unique or repeated values?
    - Mersenne Twister: 2^{19937}-1 cycle, but values repeat

Poor PRNG, a note

è l'esempio nel quale chiamare 4 volte funzione pseudo Random è peggio che chiamarne solo una! (∃ dipendenza)

➔ **Assume PRNG cycle = 2^N**

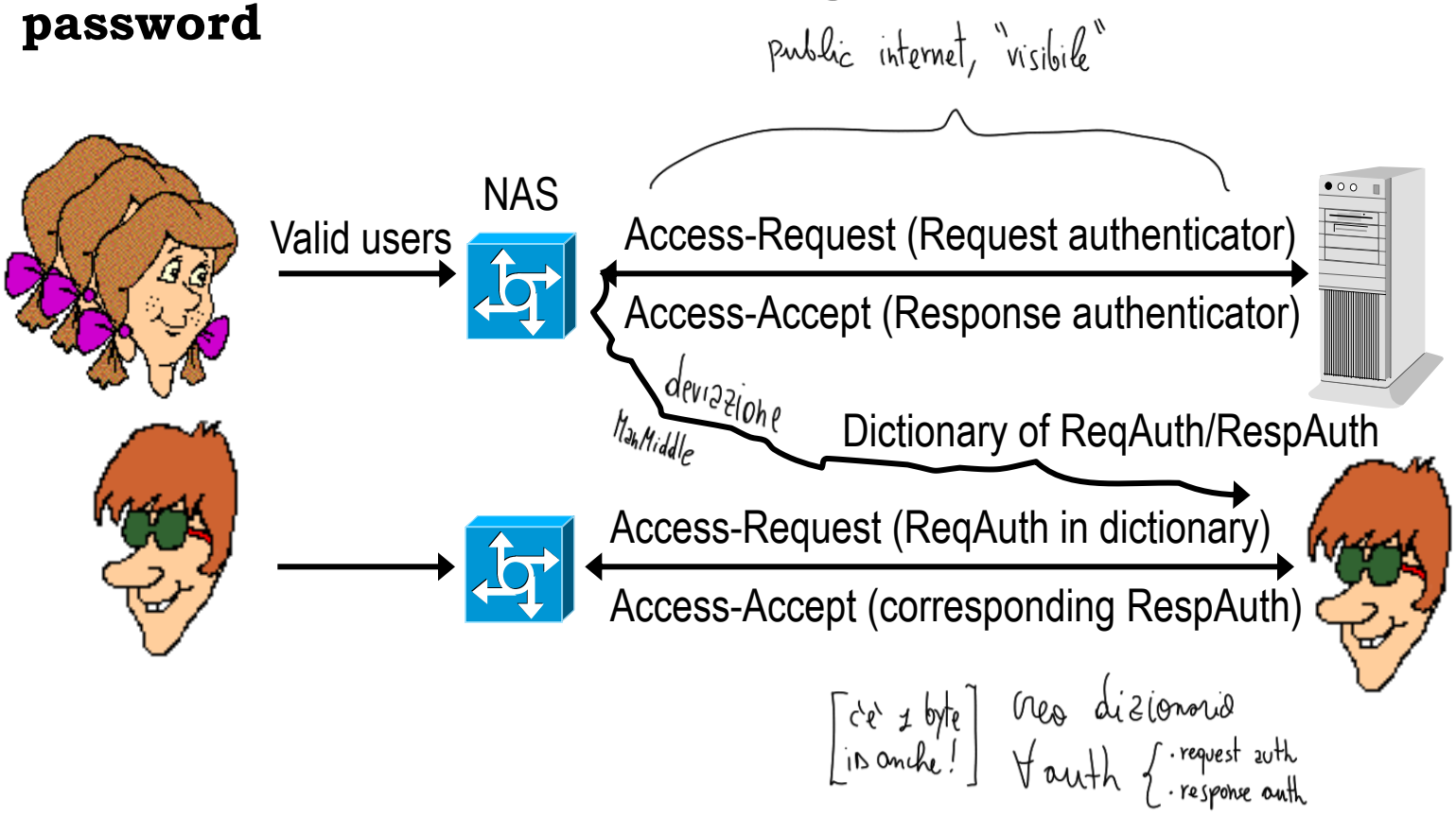
- ⇨ At best only 2^N different values!
- ⇨ Birthday attack: ½ prob with approx $2^{N/2}$
- ⇨ N=16 → birthday attack with o(250) packets!
- ⇨ C drand48 → birthday attack with o(16M) packets
  - → (not really a lot!)

➔ **Terrific (?) implementation idea:**

- ⇨ Start from 32 bit random number
- ⇨ Generate 128 bit authenticator by appending 4 subsequent 32 bit random numbers
- ⇨ Cycle: reduced to 30 bits!!
  - → Birthday attack: o(32000) packets!

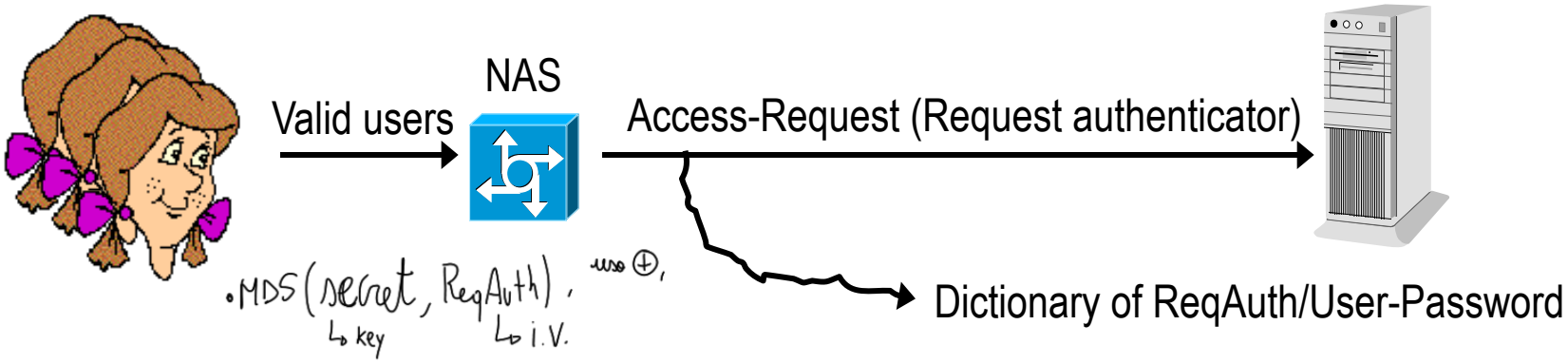
poor PRNG implementations
Replay Attacks

➔ **authenticate/authorize an illegal user with no valid password**



poor PRNG implementations
Attack to Customer passwords / 1

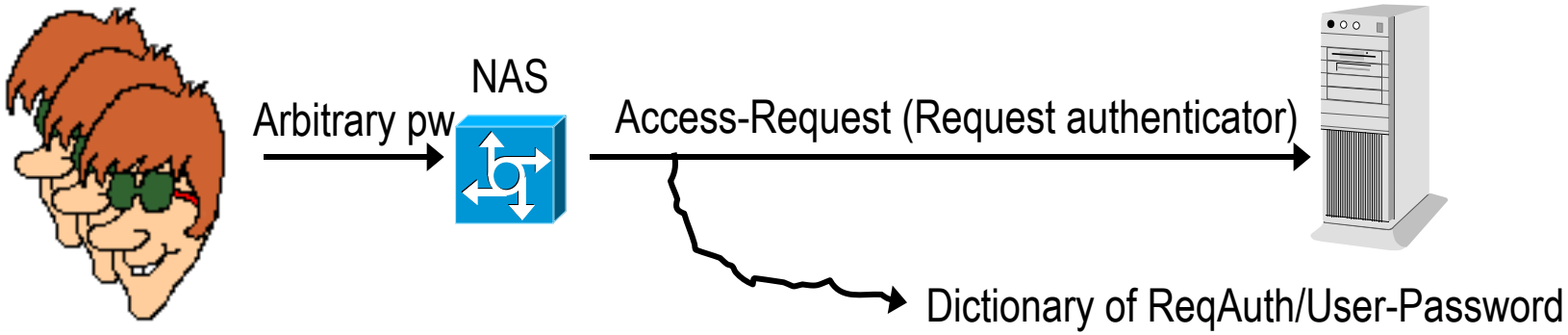
Passively monitor the network traffic allows to build a dictionary of Request Authenticators and the associated (protected) User-Password attributes



- ➔ **Repeated Request Authenticator observed**
- ➔ **XOR previous user-password with new user-password**
  - ⇨ From different users
- ➔ **Result: since ReqAuth is the same**
  - ⇨ (user-password #1) XOR (user-Password #2) =
    = [pw_user1 XOR MD5(secret,ReqAuth)] XOR [ps_user2 XOR MD5(secret,ReqAuth)] =
    = pw_user1 XOR pw_user2 (pw uguale, passo `capirle`)
- ➔ **BUT passwords from different users differ in length:**
  - ⇨ last characters of longer password are put in clear!!
  - ⇨ Password sizes are known!!
  - ⇨ Low-entropy passwords help disclosing them!

poor PRNG implementations
Attack to Customer passwords /2

ACTIVELY submit passwords chosen by attacker to add known passwords to the dictionary of Request Authenticators and the associated (protected) User-Password attributes



➔ **No need to guess right!!**
Dictionary built using rejected attempts!

Lessons learned from RADIUS

- **Do-it-all-in-one does not pay off**

- **AAA protocols should NOT implement their own security mechanisms**
  - carefully structured attacks break down RADIUS "simple" (?) approaches
  - And MD5 broken in 2005 (Wang and Yu)
    - MD5 hard to upgrade in radius (requires complete change)

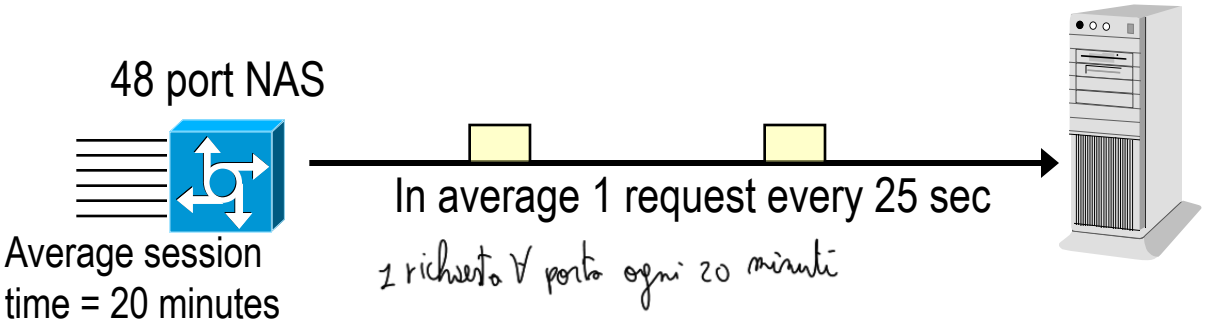
- **Current trend: rely on an underlying security layer**
  - RADSEC = Radius over TLS; RADIUS over IPsec
  - DIAMETER approach: mandatory usage of IPsec
  - Easy to say, much harder to deploy (complexity, cost, extra processing)

AAA evolution: beyond RADIUS

RADIUS today

- **Radius initially deployed to mainly support dial-in PPP users and terminal login users**
- **Today, RADIUS = de-facto standard for AAA**
  - Universal support from device vendors
- **But severe functional limits**
  - Scalability: today much larger customer base "size"
    - From a few thousands of users, as in first ISPs
    - To several M users as in cellular operators and national-level ISPs
  - Extensibility:
    - new technologies
      - Wireless, DSL, 3G, Ethernet, …
    - many more service scenarios
      - Mobile-IP, roaming, carrier-grade accounting, etc
      - new authentication mechanisms
  - Interoperability: lot left unspecified, and dealt with in proprietarily
    - E.g. failover, load balancing, server selection, etc

A note on scalability



➔ **What if 10.000 of such NAS?**
- ⇨ 400 access-requests/second
  - → 3.2 mbps server load if request averages 1KB
- ⇨ Add 400 account-requests/second
- ⇨ Add delivery of accounting records

➔ **Some RADIUS implementations may be unable to manage all this load with no packet loss!**

➔ **Finally, add load peaks** (se ho congestione) [non voglio UDP]
- ⇨ when NAS reboots (e.g. after power failure)
  - → Many accounting records back-to-back
  - → Remote users log-in simultaneously

Conclusion: AAA servers (and their networks) may experience CONGESTION!!
As well as packet drop!!

Ongoing evolution in IETF

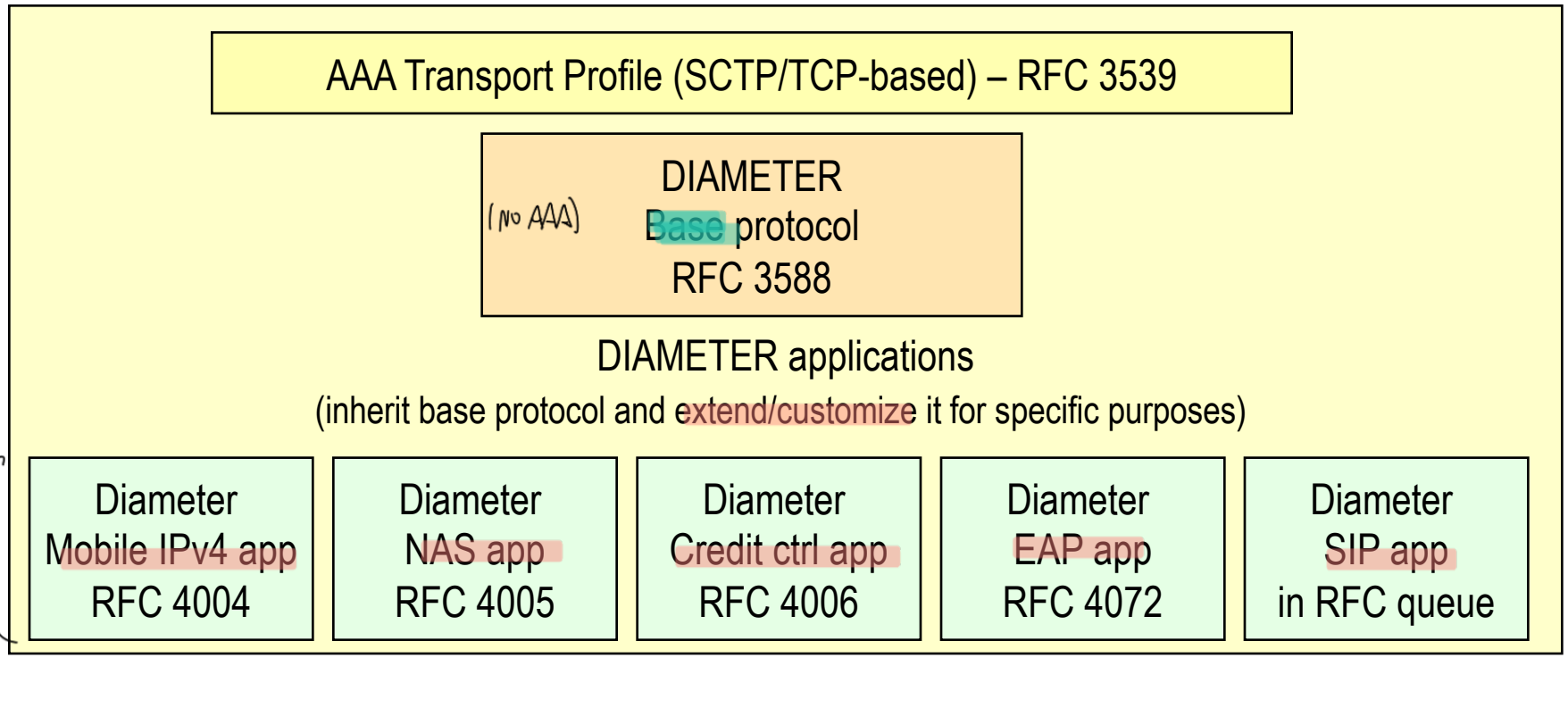
- **Diameter**
  - Started on dec 1998
  - Now completed, activities moved to DiME (Diameter Maintenance and Extensions WG)
    - First WG email on jan 2006
    - Planned actions: support of MIPv6; QoS; AAA in IP telephony (SIP) and in Local Area Networks (VLAN)
- **RADext**
  - Started on august 2003
  - RADIUS extensions with mandatory backward compatibility
    - Planned actions: AAA in IP telephony (SIP) and in Local Area Networks (VLAN), pre-paid support
  - No transport (still UDP) and Security enhancements
- **RADIUS/Diameter compatibility**
  - Goal of both WGs!!

Why "duplicating" the work?

- **RADIUS:**
  - Massive work on RADIUS deployment
    - → De-Facto standard, heavily integrated in business/ISP domains
  - Limitations can be circumvented with "small" ad-hoc extensions
- **Diameter:**
  - Brand new protocol
    - → Though somehow backward compatible, neverteless it changes lots of basics
      - » Transport, Packet format, philosophy, …
  - Much more powerful
    - → but also muuuuch more complex
  - Perfect choice in new (emerging) scenarios
    - → Most notably 3G mobility world
- **Uncertainty on what will be THE future AAA protocol, and IF there will be ONE dominant protocol: keep both!**

Diameter: whole picture

≈ protocollo object-oriented



A very complex specification! Indeed, DIAMETER = 2 x RADIUS ☺
We will NOT enter into ANY detail…

Diameter improvements at a glance /1

➔ **reliable transport**

» Versus RADIUS unreliable UDP and "proprietary" retransmission

⇨ Reliability essential in accounting

→Packet loss = money loss!

⇨ SCTP preferred (otherwise TCP)

→One persistent connection between client & server

→Packets pipelined in this single connection

SKIP

➔ **Standardized error and fail-over control**

» Versus RADIUS implementation dependent approaches

⇨ Error control and retransmission functionalities at the application level

⇨ Duplicate detection

→Especially important in accounting

⇨ Application-layer watchdog: periodic packets devised to understand when Diameter peer fails

→Default: 1 message every 30 +/- 2 seconds (artificial random jitter added to avoid syncronized arrivals)

→May be decreased to 6 +/-2 seconds

→Timer reset when other packets arrive (i.e. watchdogs used only when no other packets arrive)

Diameter improvements at a glance /2

- **Extension of functional limits**
  - 24 bit AVP filed (Attribute-Value-Pair) versus 8 bit attribute field (exhausted)
  - No more 8 bit ID (limits to 256 packets on the fly from client to same server – not scalable), but 32 bit (Hop-by-Hop identifier)
  - Duplicate detection (E2E identifier and T flag)
  - Capability negotiation: mandatory/non-mandatory AVPs (flag M) allow to restrict reject only to serious lack of capabilities
    - » Versus radius "reject" answer if request attributes not supported

RADIUS | code | id | len | auth | AVPs

Diameter header

Version: 0x01	Message length (3B)
R P E T res-flags	Command-Code (3B)
Application-ID (4B)	
Hop-By-Hop Identifier (4B)	
End-To-End Identifier (4B)	

differenziati

Flags:
R: 1=request, 0=answer
P: proxiable (specifica se il msg passa per intermediario)
E: this is an error message
T: potentially retransmitted message
(in TCP ha seq. num. qui no).

AVPs

AVP code (4B)	
V M P res-flags	AVP length (3B)
Vendor-ID (optional, 4B) "overload" il codice	
…… DATA ……	

Flags:
V: vendor-specific bit: vendor ID follows, code is from vendor
M: mandatory bit: reject if this AVP unsupported
P: privacy bit: need for e2e encryprion (è prot. interno)

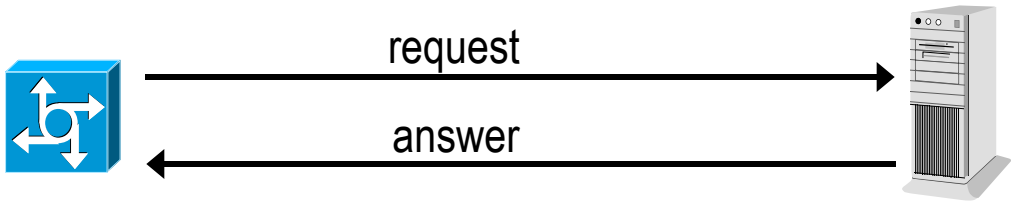
* NON posso avere unica versione, posso NON capire i msg nel pkt Diameter. M=1 importante, "devo capirlo", rifiuto pkt/msg errore, M=0 no.

Diameter improvements at a glance /3

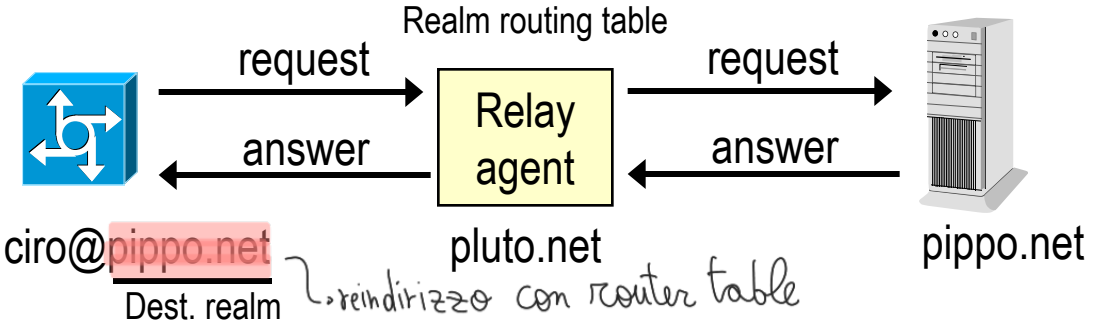
- **Peer discovery, configuration, capability detection**
  - » In RADIUS, clients are manually configured
  - ⇨ Diameter uses specific IETF protocols
    - →(Service Location Protocol version 2)
  - ⇨ C/S exchange capabilities when they set-up a transport connection
    - →Identity, diameter version, security mechanisms, supported Diameter applications
- **Supports unsolicited S→C messages**
  - » Versus RADIUS pure client-server paradigm
  - ⇨ Allows unsolicited abort/disconnect, reauthentication/reauthorization,…
- **Very detailed management of intermediate entities**
  - » Versus sloppy specification in (early) RADIUS
  - ⇨ Clear distinction between hop-by-hop and end-to-end
  - ⇨ End-To-End protection mechanisms
    - » RADIUS can only deal with Hop-by-Hop
  - ⇨ Explicit specification of Proxies, Redirects, Relay agents
    - →Roaming support
    - →Routing (!!) support

Diameter agents operation

No intermediate agent

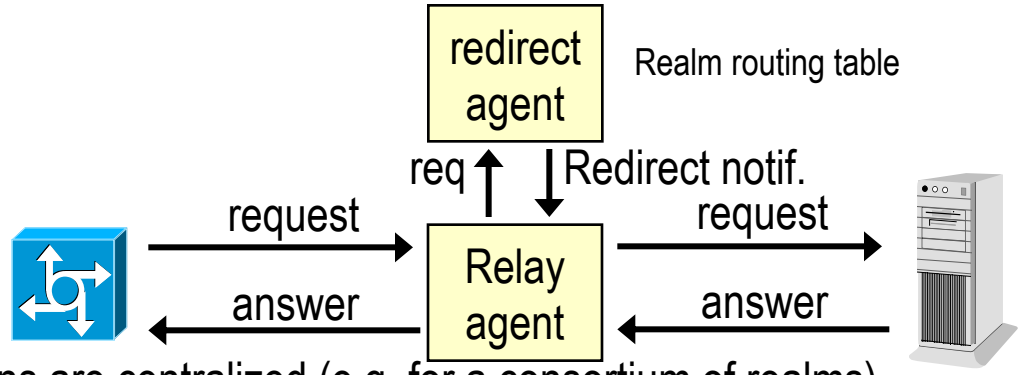


Relay agent: accept request and routes it to the proper server based on info contained In the message (destination realm)



Proxy agent: as relay but modifies message (hence e2e message authentication broken)

Redirect agent: provide routing decision on incoming request but does not actually route request (returns redirect)



Useful when Diameter routing decisions are centralized (e.g. for a consortium of realms)
Typical usage: individual relays default-route to redirect agent