Authentication over insecure networks

- User account repositories: LDAP/AD
- Password authentication standards: PAP, CHAP
- Radius

Chapter 16.1 and 16.4

2018 Sept:	Attackers Access Data of Up to 90 Million Facebook Users
2018 Dec:	New York Times Discovers Facebook Sharing User Data Without Permission
2019 March:	Up to 600 Million Facebook Passwords Stored in Plaintext Files
2019 April:	540 Million Facebook User Records Found on Public Server
2019 April:	Facebook Uploads 1.5 Million Users' Email Contacts Without Permission
2019 July:	Federal Trade commission, FTC Imposes $5 Billion Penalty for privacy violations
2019 Sept:	Data for 419 Million Facebook Users Found on Exposed Server
2019 Dec:	Hacker Group Captures Data from 300+ Million Facebook Accounts
2020 June:	Facebook Accidentally Shares User Data with Third-Party Developers
2021 April:	Personal Data for Over 530 Million Facebook Users Leaks in Online Forum

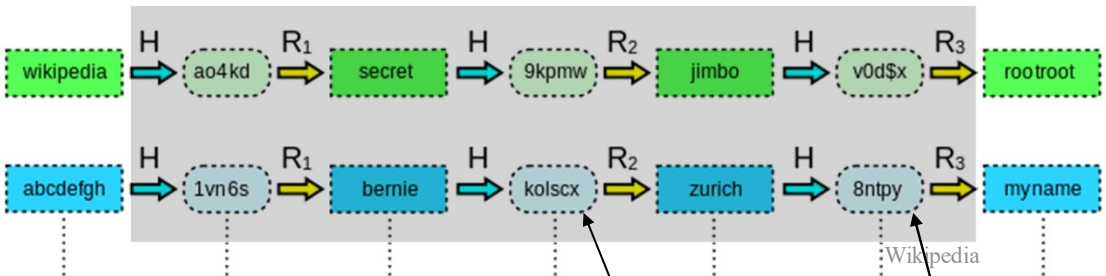
…

https://www.theverge.com/2022/11/28/23481786/meta-fine-facebook-data-leak-ireland-dpc-gdpr
https://firewalltimes.com/facebook-data-breach-timeline/

Passwords

- Physical access to a system should not give access to clear-text passwords
  - A stolen computer or access to the database may lead to massive password leaks
  - Users often reuse passwords on different systems/sites
- Passwords often stored “encrypted” with a one-way hash function:
  - hash (”my secret password”) → Xp7LiOpjfHrl3x9opJ!0Jm
- If the database is stolen, simple passwords can be revealed:
  - Hash all words in dictionaries, forward, backward, upper/lower case, add numbers, …
  - Include all shorter passwords: hash(”a”), hash(”aa”), hash(”aaa”), hash(”b”), …
- Pre-calculated lookup tables can be created – or downloaded
  - Rainbow tables saves space
  - Mixing password with a “salt”, a random string, makes it stronger: hash(password, salt) requires one lookup table per salt

Rainbow tables (hash chains)



- A reduction function R maps hashes back to passwords (any function returning text of a defined length)
- Store only the first and last value of a chain of length k
  - Sort on last value.
  - Chain length is a tradeoff between time and storage (length=1 is a traditional lookup table)
  - Tables can be built for at least all 14-character combinations
- Mitigations:
  - Use salts – each server is uniqe
  - Key stretching: Hash *n* times with salt before storing (n > 1000)
  - Key strengthening: Use random salt of *m* bits where both attacker and verifier needs to search for it (time consuming)

- Using the table:
  - For a given password hash, apply R and see if it matches a "last value" (use sorted list for speed)
  - If not, maybe the previous hash entry matches? From the result of the previous R, apply H and then again R
  - Repeat max k times
  - If match: recreate the full chain from the beginning to find the password (since only first and last value is stored)
  - It may be a false alarm, R is not collision resistant
- Duplicate entries may occur at many places in the table
  - Makes rainbow tables less space efficient
  - A sequence of different reduction functions may be better? $R_1, R_2, R_3, \ldots, R_k$

Extracting the hashes from Windows 10

It is simple – tools exist:

1. Save the HKEY_LOCAL_MACHINE System and SAM hives (databases) to two files
   - SAM = Security Account Manager
   - Administrative privileges are needed
2. Download *mimikatz* or a similar freeware tool from Github
3. Shows account names and corresponding hashes of the passwords
4. Now just crack the hash… (next slide)

```
mimikatz # lsadump::sam sam3.hiv system.hiv
Domain : EC2AMAZ-24TU91P
SysKey : 505d8aac5a6cd368b6fe7ec1be2c08e7
Local SID : S-1-5-21-1743574332-290634130-2548908371

SAMKey : 7509fe8f5cdf2281fe94347e13a793a0

RID  : 000001f4 (500)
User : Administrator
  Hash NTLM: daa416a72d3e83bae2d3434493648407

RID  : 000001f5 (501)
User : Guest

RID  : 000001f7 (503)
User : DefaultAccount
```

Password cracking using GPUs

IBM's X-Force Red team have built a system for penetration testing:

- 8 Nvidia GTX 1080 FE cards [2018]
  Each card has 2,560 GPUs (cores) → 20,480 cores @ 1.7 GHz
- Performance (using *Hashcat*):
  - SHA1: 66.1 GH/s
  - NTLM: 334 GH/s = all possible 8-character passwords in 12 hours
  - There are approx. 175 000 English words in use today…

- Nvidia GeForce RTX 4090 [2023]
  - 16,384 cores @ 2,5GHz (faster than the 8 IBM cards above!)
  - 24GB memory
  - Price approx. 20 kSEK

LDAP

User account database access

LDAP

- Light-weight Directory Access Protocol
  - First version 1993
  - Version today is v3 (LDAPv3)
  - RFC 4510 to 4519
- Protocol is used to access directory listings (information about users)
  - Passwords, roles, organization, mail address, phone number, …
  - Database format is often X.500
- Communicates in clear, should be secured with TLS
  - Just a protocol for database access

- Linux systems often include OpenLDAP
  - Client libraries plus a server
  - Can be used by various applications: Servers, web servers, mail clients, …
- Microsoft's Active Directory (AD) server supports many authentication protocols:
  - LDAP, Radius and Kerberos

LDAP

- LDAP is a protocol
  - OpenLDAP, AD, etc. are systems/implementations using the protocol
- Typical operations: get, add, delete, modify, search, compare, extend
- Each entry consists of a set of attributes
  - Each entry has a unique name: **dn** (Distinguished Name of entry/record)
  - Attribute = name and associated value(s)
- Example in LDAP Data Interchange Format (called **ldif**):

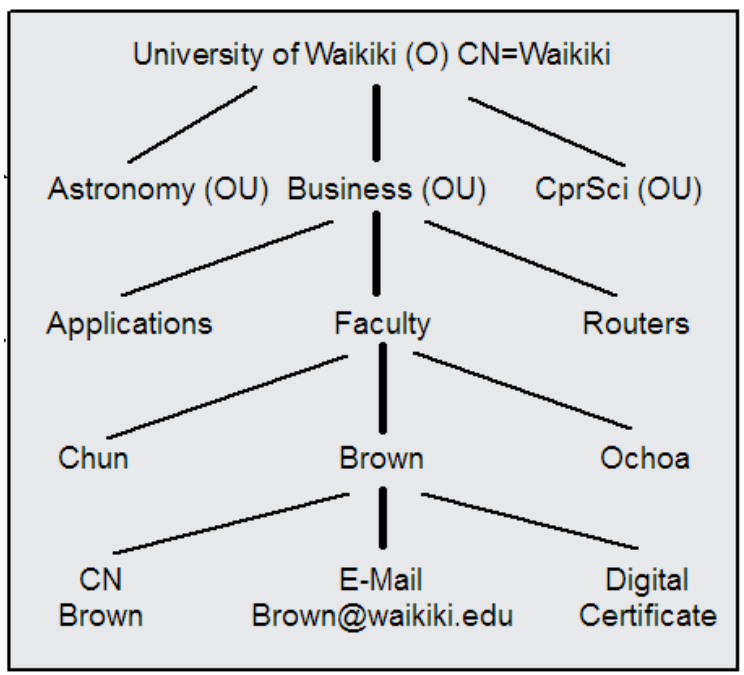
```
dn: cn=John Smith, dc=company, dc=com  # distinguished name of entry/record
sn: Smith                              # Attributes follow. sn = surname
givenName: John                        # …
userPassword: fjjf3¤#jfkl
mail: john.smith@company.com
telephoneNumber: 1 555 123-4567
…
```

cn = common name = name of the object
dc = domain component = parent name, i.e. where in the tree the object fits

LDAP Directory Server Organization

GET E-Mail.Brown.Faculty.Business.Waikiki

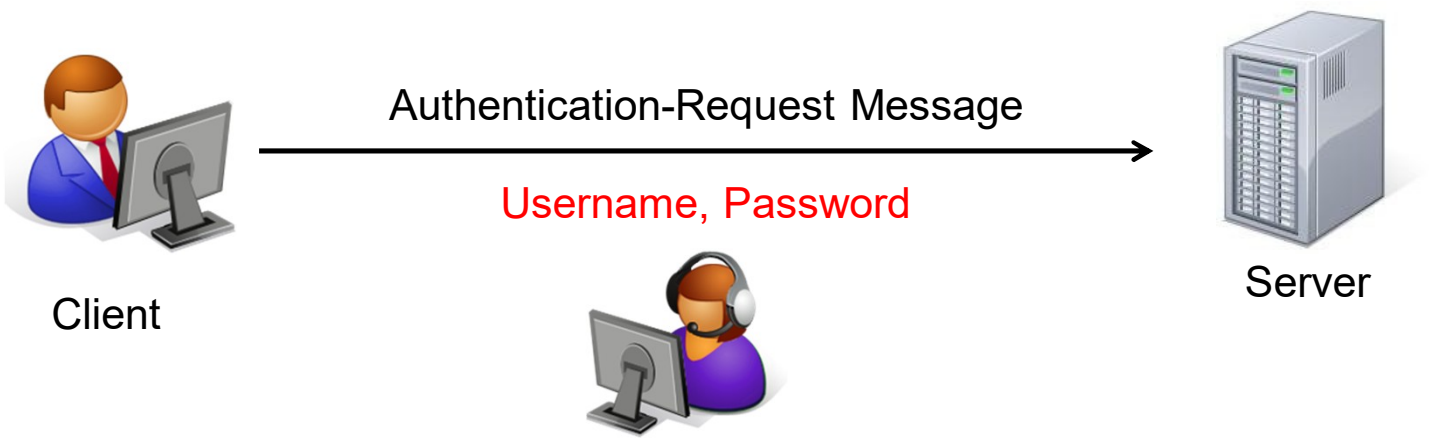
Brown@waikiki.edu



Standards for Password Authentication

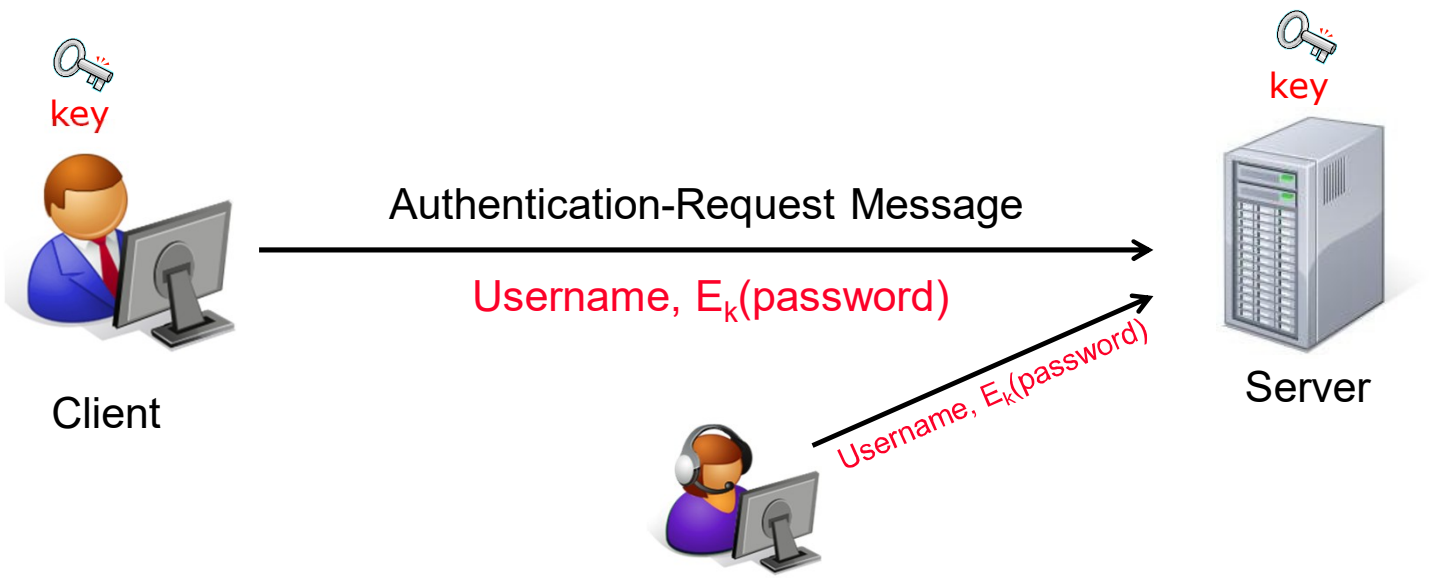
Sending passwords over networks

PAP: Password Authentication Protocol (RFC 1334)



Poor Security: Usernames and Passwords are sent in clear.
Can only be used if connection is encrypted

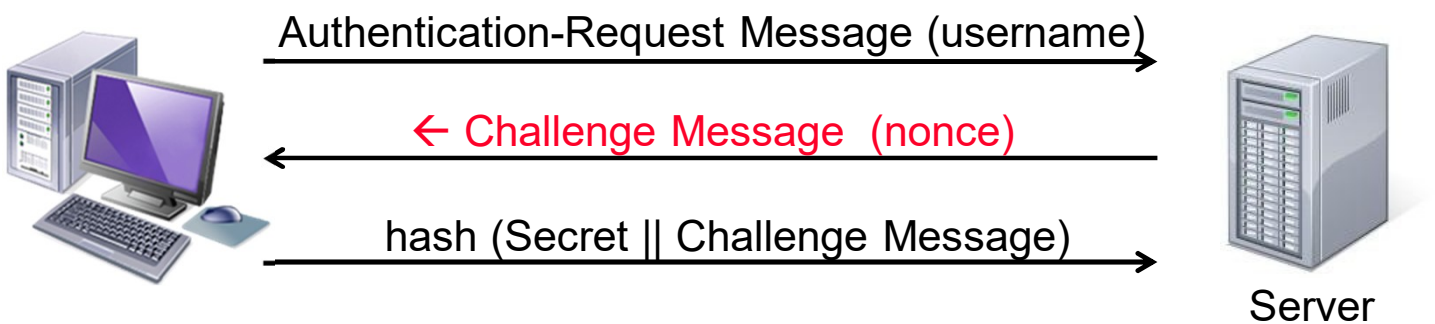
Better to send password encrypted?



Can be replayed:
E_k(password) is equivalent to a password, just harder to read

CHAP: Challenge Handshake Authentication Protocol

[RFC 1994]



Client computes hash of challenge message and secret.
Server performs the same computation and verifies the response.
Hash is better than encryption – fast, no politics, same security.

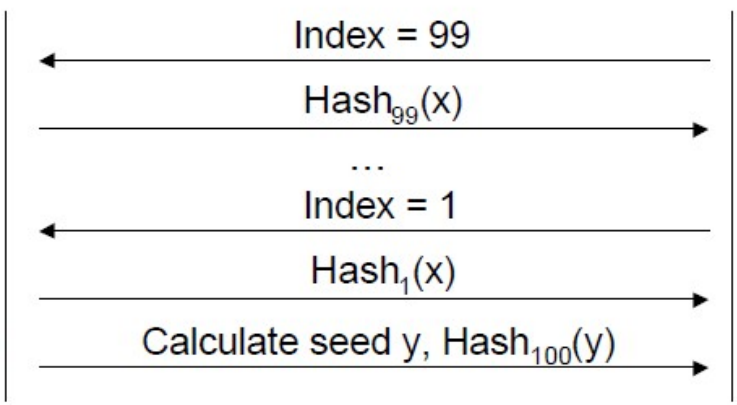
Since the server must be able to do the same operation on the challenge, the secret is normally stored in cleartext on the server.

One-time passwords: S/Key algorithm

$$Hash_n(x) = \underbrace{Hash(Hash(\ldots Hash(x)))}_{n \text{ times}} = Hash(Hash_{n-1}(x))$$

Knows x

User only stores x.
All hashes can be computed.



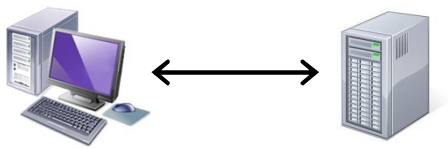
$Hash_{100}(x)$

Server only stores last hash.

$Hash_{n-1}$ received next time, checked against $Hash_n$ which is then stored.

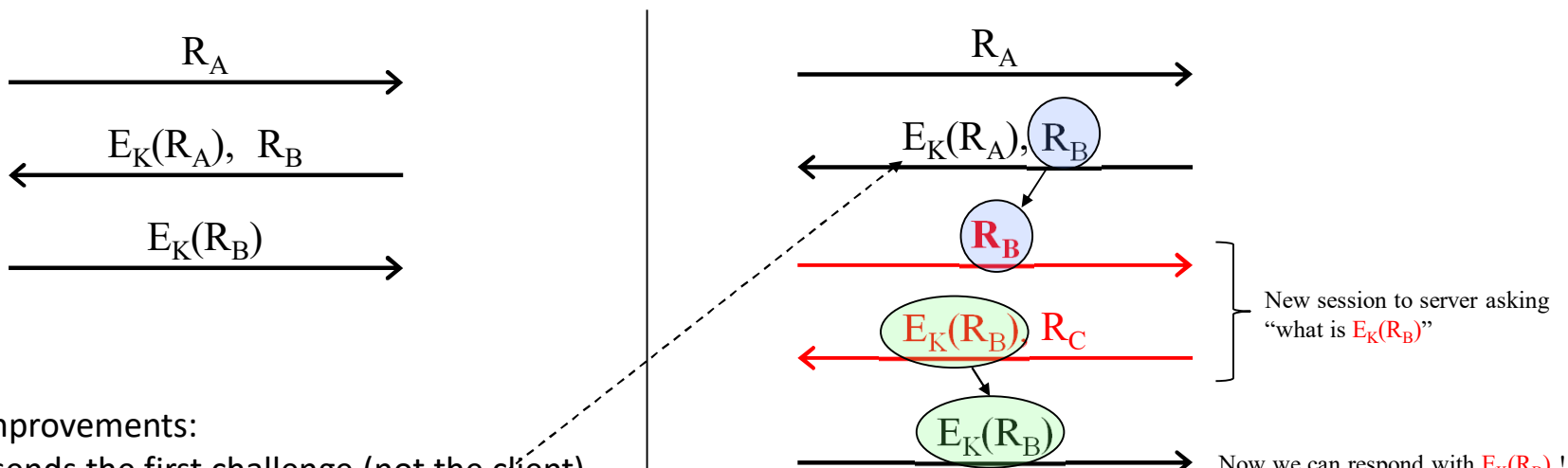
- Almost a Scratch list: requires brute force to crack
- Invented by Leslie Lamport (1980's)
- Client must have processing power for authentication
- Printed list with hashes also possible

Reflection attacks



It is not trivial to always foresee problems in protocols

This protocol may seem to work:



Possible improvements:

- Server sends the first challenge (not the client)
- Include a nonce or time stamp in the calculations
- Server responds with $E_K(R_A, R_B)$

R_A = random number (nonce) from A
K = shared key

ISO 9798: Entity authentication

ISO has standardized some authentication methods:

- ISO 9798-2 → Symmetric algorithms

Part 2: Symmetric-key Cryptography	
9798-2-1	One-pass unilateral authentication
9798-2-2	Two-pass unilateral authentication
9798-2-3	Two-pass mutual authentication
9798-2-4	Three-pass mutual authentication
9798-2-5	Four-pass with TTP
9798-2-6	Five-pass with TTP

Next slide

- ISO 9798-3 → Digital signature techniques
- ISO 9798-4 → Using cryptographic hash functions, MACs Next slide

(TTP = trusted third party)

ISO 9798-2-4 protocol

Three-pass mutual authentication with symmetric key encryption

$R_A \parallel Text_1$ →

← $Text_3 \parallel E_K (R_B \parallel R_A \parallel ID_A \parallel Text_2)$

"I can encrypt this number with our key, can you recover R_B?"

$Text_5 \parallel E_K (R_A \parallel R_B \parallel Text_4)$ →

Both A and B knows the other party is alive. Responses contain both R_A and R_B to show that they belong to the same session.

ISO 9798-4-3 protocol

Two-pass mutual authentication with cryptographic hash functions

$TN_A \parallel Text_2 \parallel hash_K(TN_A \parallel ID_B \parallel Text_1)$ →

"I can hash this number with our key"

← $TN_B \parallel Text_4 \parallel hash_K(TN_B \parallel ID_A \parallel Text_3)$

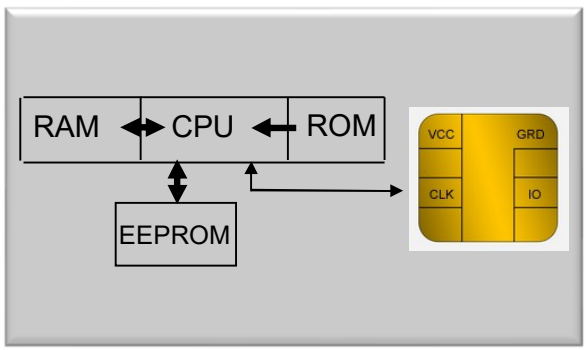
R = Random number (nonce)
TN = Time stamp or sequence number never reused!
ID = Identity of the party
K = shared key
Texts are optional and their use is unspecified

Two factor authentication

- Token cards implement two-factor authentication
  - PIN code entered on device keyboard
  - Secret, device dependent key stored inside device
  - Challenge from server may also be entered
  - Server may require PIN + response from device
- Output may be based on time (synchronized clocks)

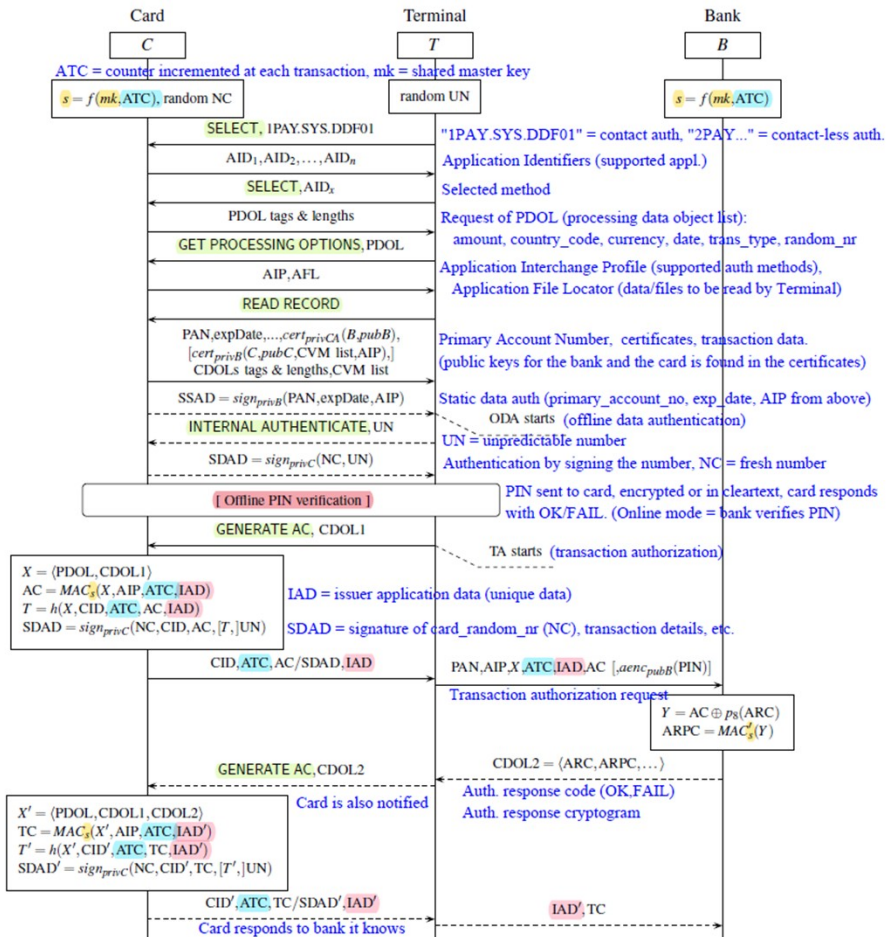
Smart Cards and USB tokens

- Contains memory (RAM, ROM) and a microprocessor
  - ROM for the program, RAM for variables, etc.
- 8, 16 or 32-bit Risc processor, 25-30 MHz (e.g. ARM)
- Memory typically 32-500 kB ROM, 32kB EEPROM and 1-8 kB RAM
- Can have hardware for AES encryption
- Uses serial communication to card readers (ISO/IEC 7810 and 7816)
  - 9,600 to 100+ kbit/s
- Operating system in ROM decides what functionality to offer
- Private keys can be stored in EEPROM
  - Card can perform operations (sign challenges, transactions, etc.)
  - Key never leaves the chip
  - PIN code can be required by the OS before it performs any operations



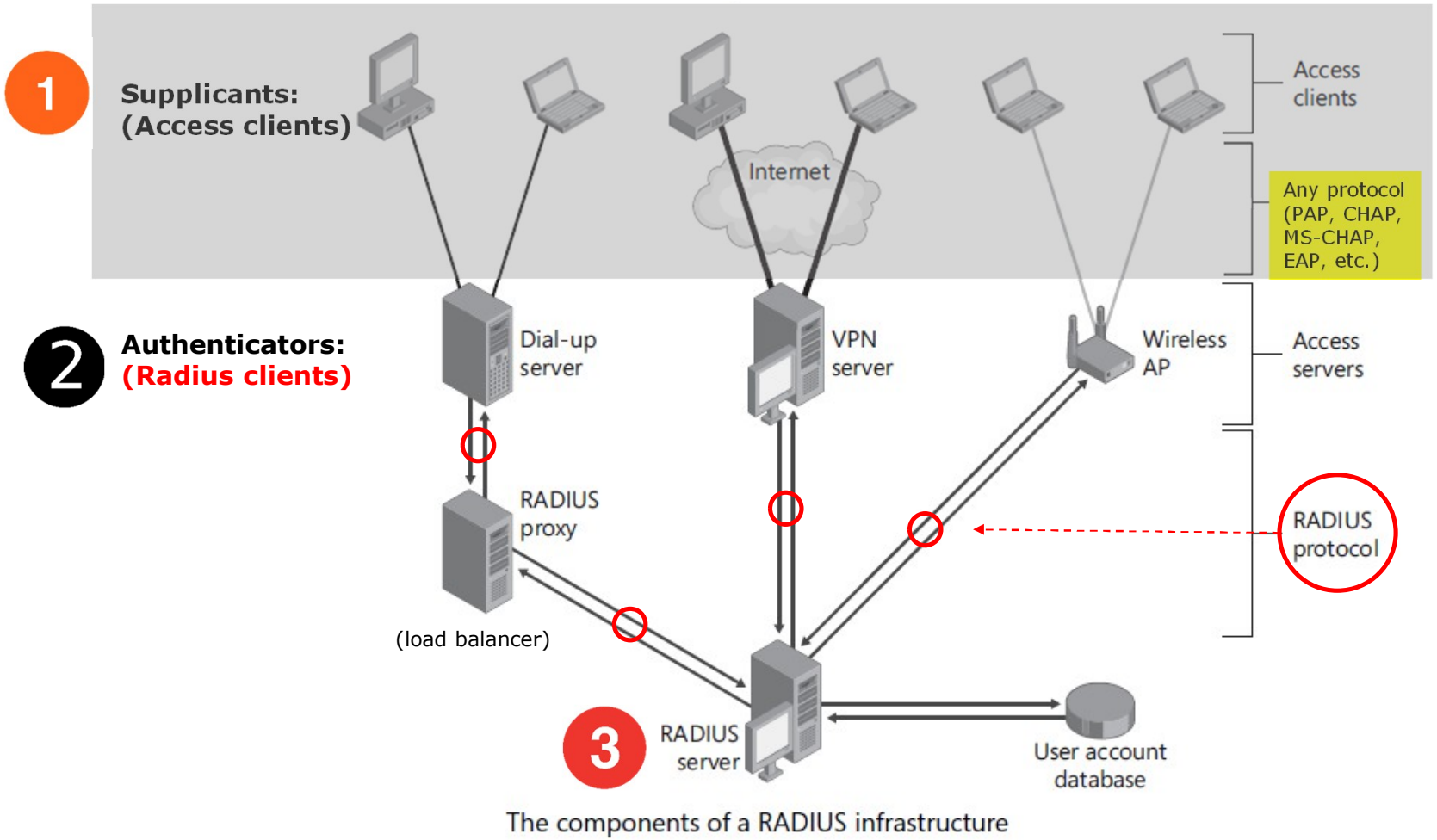
The EMV standard for credit card payments

- Standard for communication between a **Credit card**, a **Terminal** and the **Bank**
  - EMV = Europay, Mastercard, Visa
- Both contact and contact-less authentication supported
  - This slide shows contact-based authentication
  - Complex standard, more than 2,000 pages
  - Many vulnerabilities have been found
- Main steps:
  - Selection of algorithms
  - Exchange of certificates
  - Transaction signing by card
  - Involving the bank for authorization



https://arxiv.org/abs/2006.08249

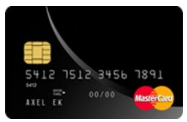
Radius – RFC 2865, …



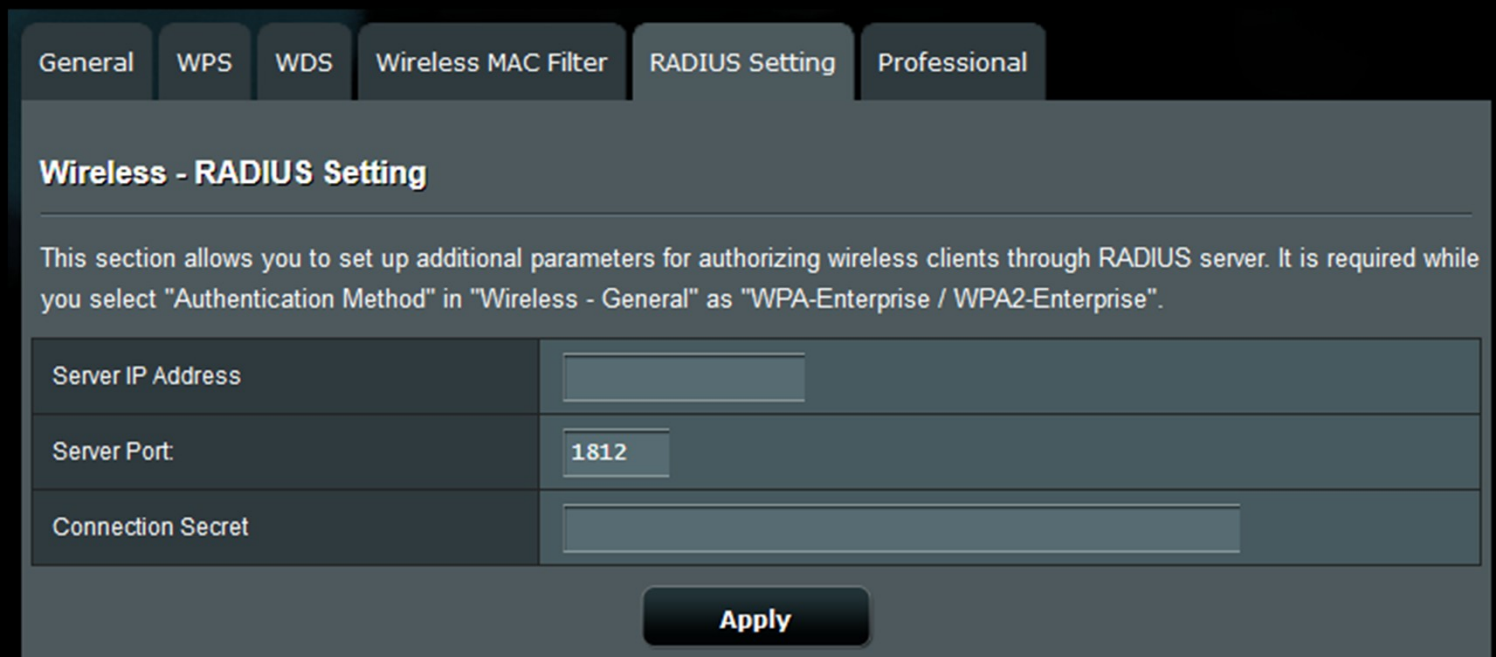
The components of a RADIUS infrastructure

Radius

- Remote Authentication Dial In User Service
  - RFC 2865, RFC 2866, …
  - De-facto standard protocol for remote authentication
  - Supported and used by almost all vendors of authentication devices
- A Radius server verifies user identities
  - Centralized administration of identities
  - Application servers do not have to keep user databases
  - Very useful for small networked devices (routers, access points, etc.)
- Often referred to as an AAA server (authentication, authorization, accounting)
  - Authentication messages are sent to UDP port 1812
  - Accounting messages are sent to UDP port 1813
- Microsoft's Active Directory server can also act as a Radius server
- The Radius client (Authenticator) and the Radius server share a common secret
  - Requests without the correct secret are silently ignored



Configuring Radius in a WLAN access point



Wireless N Access Point D-LINK DAP-2020

400 SEK

Just a cheap access point found after a quick search

Technical Specifications		
General		
Network Standards	• 802.11n wireless LAN • 802.11g wireless LAN • 802.11b wireless LAN	• 802.3/802.3i/j/u 10BASE-T/100BASE-TX Ethernet • ANSI/IEEE 802.3 NWay auto-negotiation
Device Interfaces	• 802.11n/g/b wireless LAN	• One 10/100BASE-TX Ethernet LAN port
Operating Frequency	• 2.4 to 2.4835 GHz	
Operating Channels	• ETSI: 13	
Radio & Modulation Schemes	• DQPSK, DBPSK, CCK, OFDM	
Functionality		
Operating Modes	• Access Point • Wireless Client • Bridge • Bridge with AP	• Repeater (range extender) • WISP Client Router • WISP Repeater
Antennas	• Two 5 dBi gain detachable omni-directional antennas with RP-SMA connector	
Security	• 64/128-bit WEP data encryption • WPA-PSK, WPA2-PSK • WPA-EAP, WPA2-EAP (Radius) • TKIP, AES	• MAC address filtering • SSID broadcast disable function • WPS (Wi-Fi Protected Setup)
Advanced Features	• Quality of Service (QoS): Wi-Fi Multimedia (WMM)	
Device Management	• Web-based management through Microsoft Internet Explorer 6 or higher, Firefox 3.0 or higher, or other Java-enabled browser	
Status LEDs	• Power • Wireless	• Security • LAN

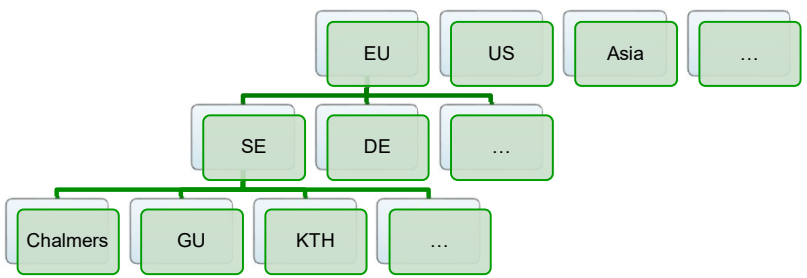
Eduroam uses Radius

What is eduroam?

eduroam (**edu**cation **roam**ing) is the secure, world-wide roaming access service developed for the international research and education community.

Having started in Europe, eduroam has gained momentum throughout the research and education community and is now available in 71 territories.

- Radius authentication done in a tree-like structure
- User = cid@chalmers.se



Eduroam, the movie: https://www.youtube.com/watch?t=14&v=TVCmcMZS3uA

Radius authentication

Supplicant

Any protocol,
e.g. WPA2

Authenticator
e.g. an access point
(shares a secret with
the Radius server)

Access request: 16_octets_random_data, ID, username, password*, message_authenticator

Access_accept/reject, ID, resp_authenticator

Radius
Server

```
XOR:
  101010…
⊕ 001100…
  100110…
```

ID = a unique number for this request – matches requests with responses

password* = MD5(shared_secret, 16_octets_random_data) ⊕ password

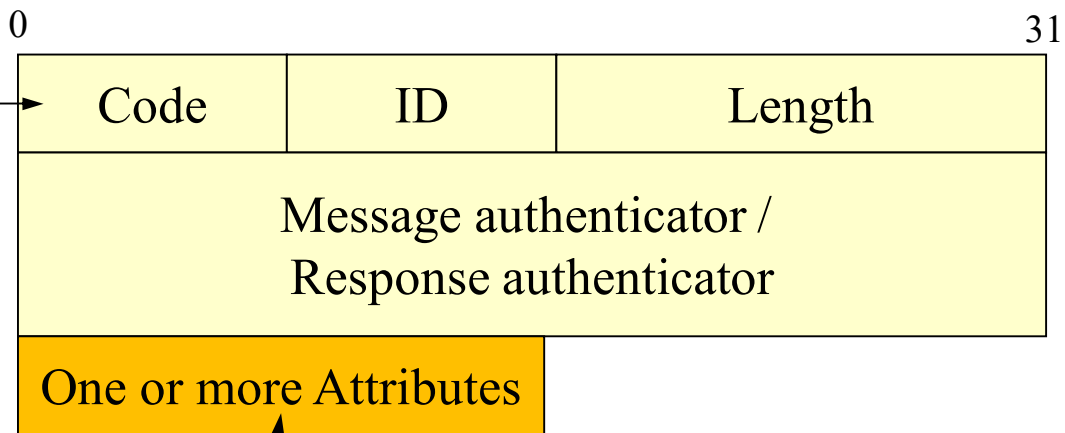
message_authenticator = MD5(packet_contents, shared_secret)

resp_authenticator = MD5(packet_contents, 16_octets_sent_by_client, shared_secret)

To prevent modification
of requests and responses

Message format, overview

Code	Description
1	Access-Request
2	Access-Accept
3	Access-Reject
4	Accounting-Request
5	Accounting-Response
11	Access-Challenge



For example
username/password*
(around 50 attributes exist)

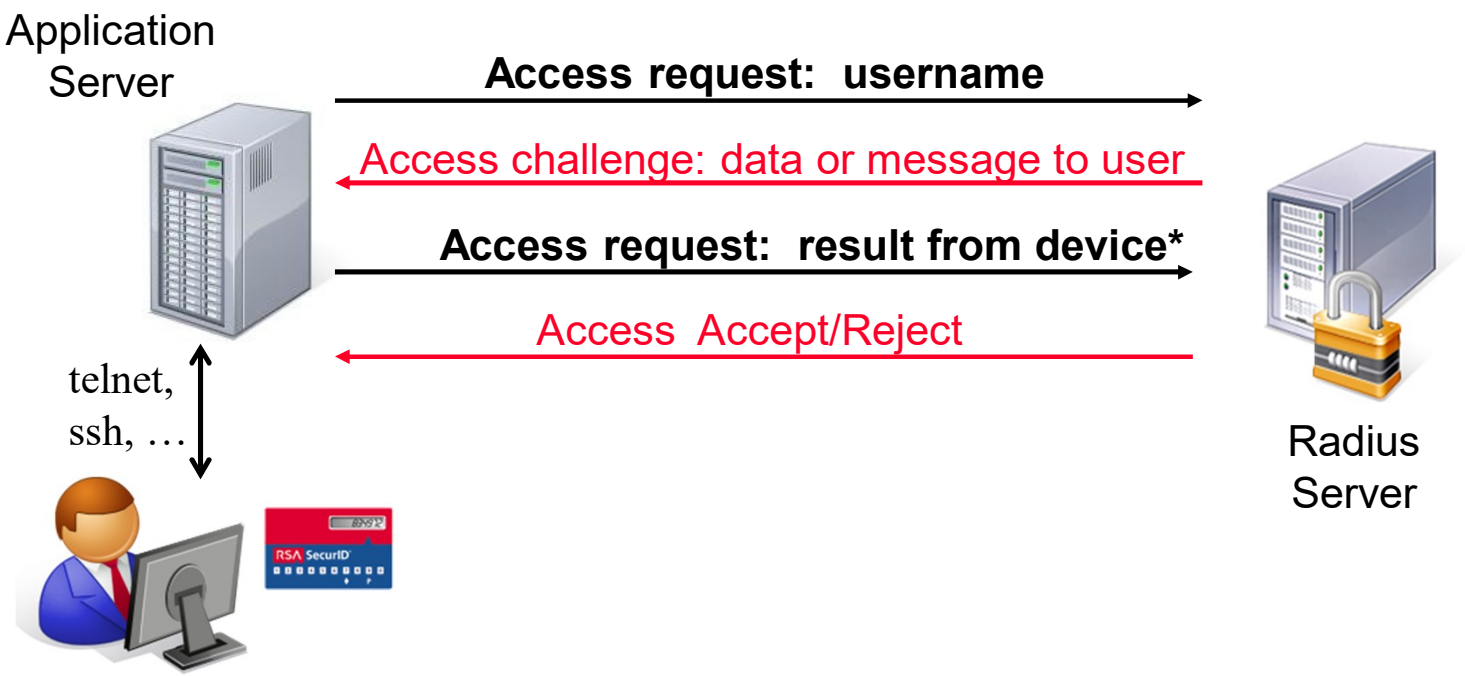
Radius – some observations

- The octets of random data *must* be random and completely unpredictable
- Failed requests are silently dropped
  - If client is not in possession of secret
  - If message authenticator is not correct
  - Unauthorized clients will not even see that the service/server exist (no replies)
- Change secret regularly to stop offline searches (see following slides)
- The protocol is slightly more complicated than outlined here

- Retransmission possible after timeout
  - Same ID reused
  - Server is stateless, redundant systems are easy to build (primary, secondary, …)
- password* allows the server to retrieve the user password in cleartext:

  MD5(shared_secret, 16_octets_random_data) ⊕ password

Radius supports challenge-response auth.



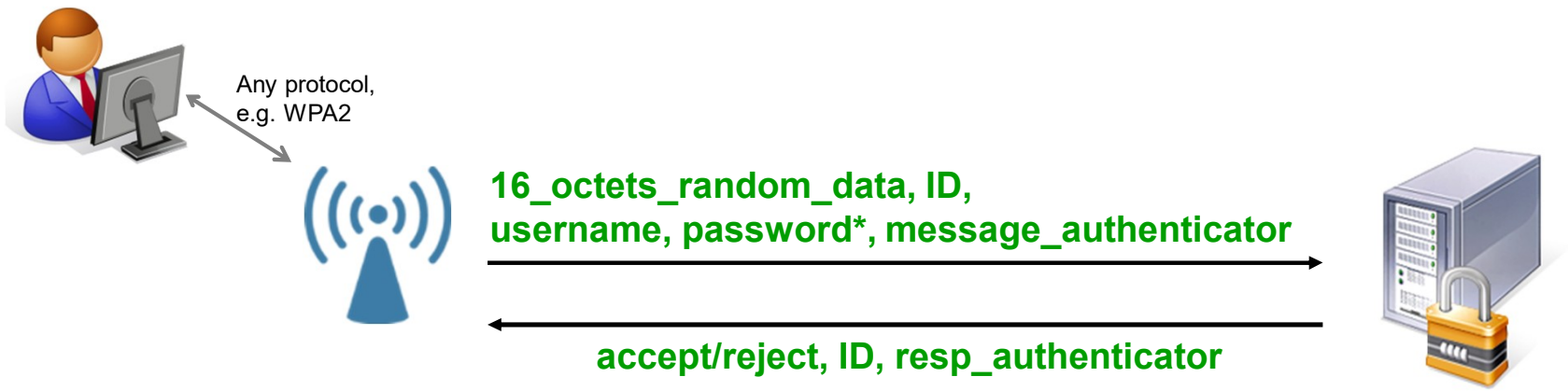
* MD5(shared_secret, 16_octets_random_data) ⊕ password

Analyzing a protocol (here Radius)

An Analysis of the RADIUS Authentication Protocol
https://www.untruth.org/~josh/security/radius/radius-auth.html

This is an exercise in analyzing a protocol

How much is sent in clear text?



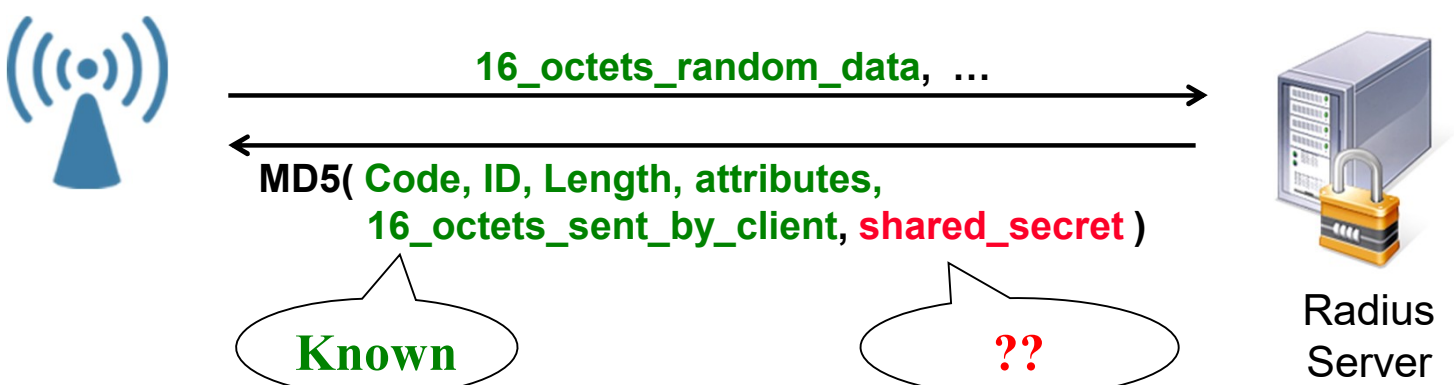
password* = MD5(shared_secret, 16_octets_random_data) ⊕ password

message_authenticator = MD5(packet_contents, shared_secret)

resp_authenticator = MD5(packet_contents, 16_octets_sent_by_client, shared_secret)

Weaknesses in Radius (1)

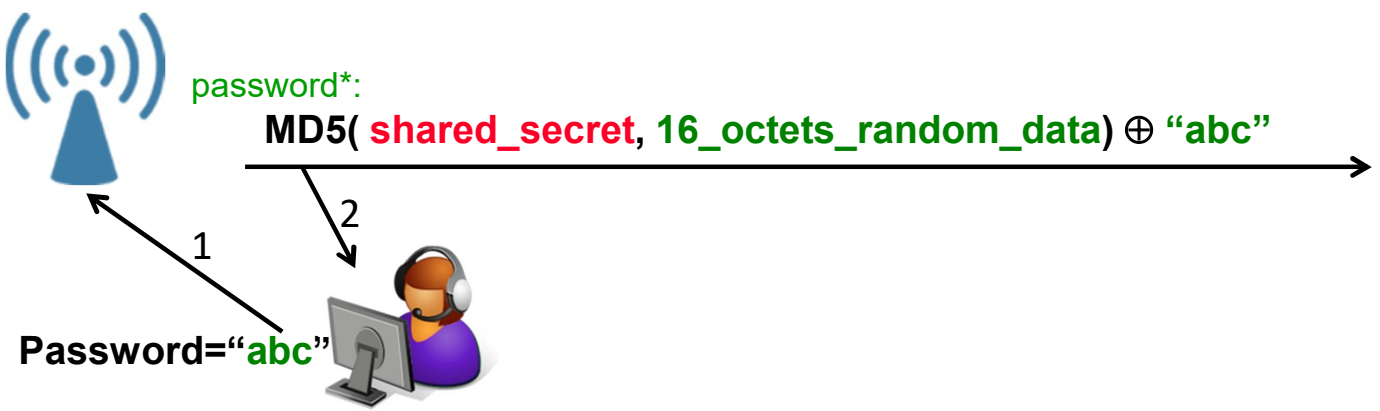
- See “An Analysis of the Radius Authentication Protocol”
- Some weaknesses even documented in RFC 3580
- [3.1] Off-line attack against shared secret using the Access-response message:



Weaknesses in Radius (2)

[3.3]

- Connect using a known password
- Capture the resulting Access-request packet (if possible…)
- We now know the hash and can launch an *off-line exhaustive search of the shared secret*
- Solution: protect the transmission to the Radius server

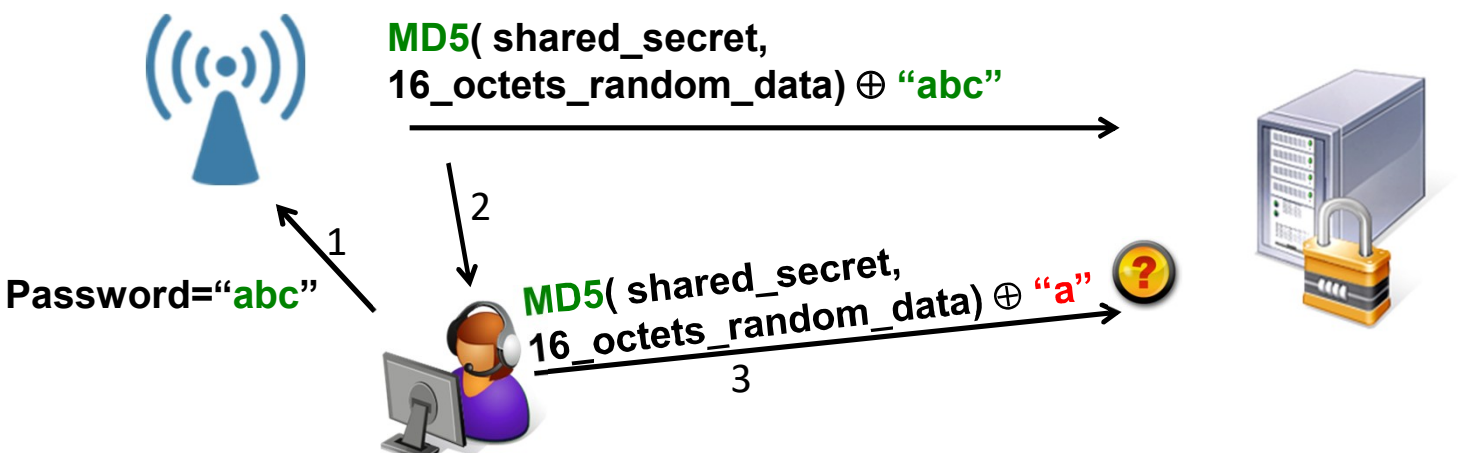


Weaknesses in Radius (3)

[3.4]

To guess another user's password:

- Connect with known password, capture the Access-request message (previous slide)
- Reuse MD5(…) with the user account we want to crack, modify until password matches
- Does not work if Radius server limits number of attempts
- Does not work if m*essage_authenticator = MD5(packet_contents, secret) which is the reason why it is present*



More Radius weaknesses

The Authenticator is the 16 octets (bytes) of random data:

[3.5.1] If authenticators are ever reused:

- Then possible to XOR two requests with each other:
  (MD5(…) ⊕ password1) ⊕ (MD5(…) ⊕ password2) = password1 ⊕ password2
- Also, if one password is longer than the other, the tail is displayed in cleartext

Takeaways:

- It is not trivial to create even simple and secure protocols for authentication
- Always ask yourself: “what is known?” and “what may be calculated?”

Radius Conclusions

- Radius is still useful, despite these problems!
- Several flaws can be solved in the implementation
  - Cryptographic pseudo-number-generator must be used when calculating random octets
- If possible, make it impossible for users to listen to the traffic between the Radius client and Server
  - Physical separation may be possible
- IPsec is recommended to be used to protect messages in sensitive environments
  - Supported by many devices and systems (Microsoft, …)
  - Necessary if Radius is used to distributed session crypto-keys
- Most practical problems are bugs in the implementations
  - This a common problem for all security solutions
  - It is not enough to just specify a protocol
  - The underlying assumptions must also be understood by the implementers

Summary

- Password authentication
  - PAP, CHAP, password handling
  - Different types of attacks, Rainbow tables, reflection attacks
- LDAP
  - Lightweight Directory access protocol
  - Often used to access user information over networks
  - Needs SSL or other mechanism to protect transmission
- Radius
  - Application to Radius server authentication protocol
  - Weaknesses
  - Protocol implementers must understand assumptions