Topic 3.1 - Authentication

Eric Casanovas

Universitat d'Andorra

2nd March, 2022

Introduction

- The authentication is the process or action of proving or showing something to be true or valid
- Possible with public key cryptography without preshared keys
- Impossible with symmetric key cryptography without preshared keys
- In internet we need to authenticate to make sure that we are talking with to who we believe

PFS problem

- PFS comes from **Perfect Forward Secrecy**
- Imagin an attacker has recorded past communications
- Later manages to get the secret key
- Can decrypt all past communications!!
- To guarantee PFS you have to use different keys for each communication

Mutual Authentication

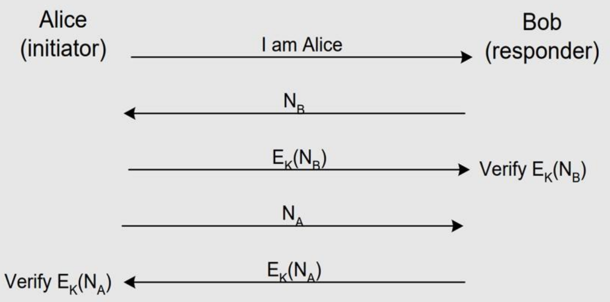
We need mutual authentication in:

- Symmetric key
- Asymmetric key
- Signatures
- Session key

Symmetric Key Authentication I

Symmetric key Authentication:

- Alice and Bob share a secret key K, and N is a nonce.

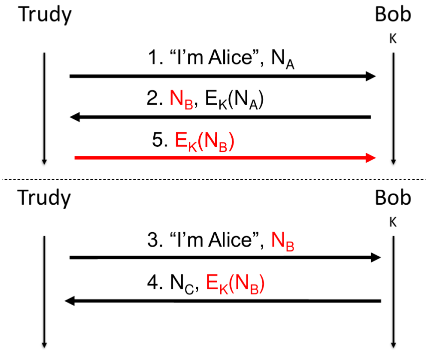


Symmetric Key Authentication II

- Do you think it is secure?

Symmetric Key Authentication III

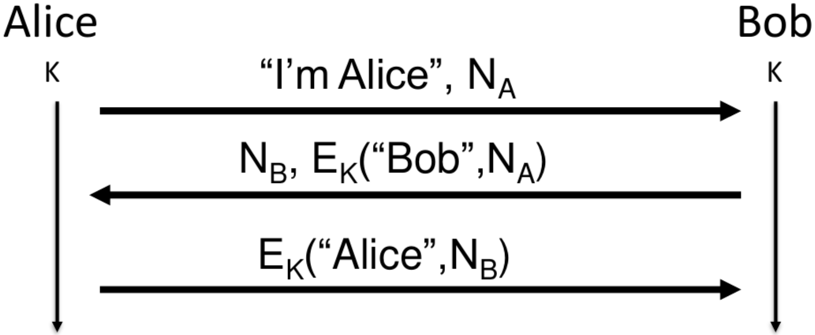
- Do you think it is secure? NO



Symmetric Key Authentication IV

- An attacker can impersonate Alice or Bob using a replay attack
- Trudy can impersonate Alice or Bob if N is repeated
- If N repeated Trudy can use $E_k(N_b)$ to impersonate Alice as E_k is public
- To increase security we have to think about a solution:
  - Don't repeat numbers (Use random big numbers as nonce)
  - Add a nonce to the encryption

Symmetric Key Authentication V



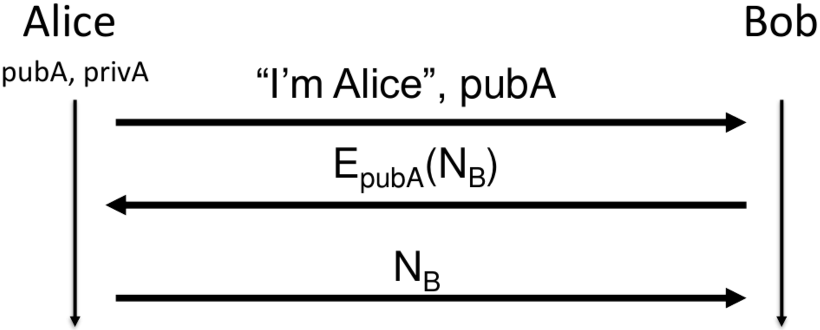
- Is this better? YES!

Symmetric Key Authentication VI

- The video

Asymmetric Key Authentication I

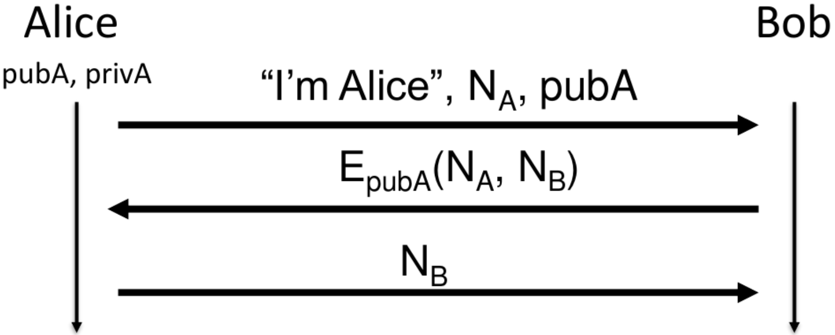
Asymmetric key authentication:



Asymmetric Key Authentication II

- Secure?
- No
- Bob can be Trudy!
- No mutual authentication, just Alice authenticated
- Use different keys for different purposes
- But we can add a nonce

Asymmetric Key Authentication III

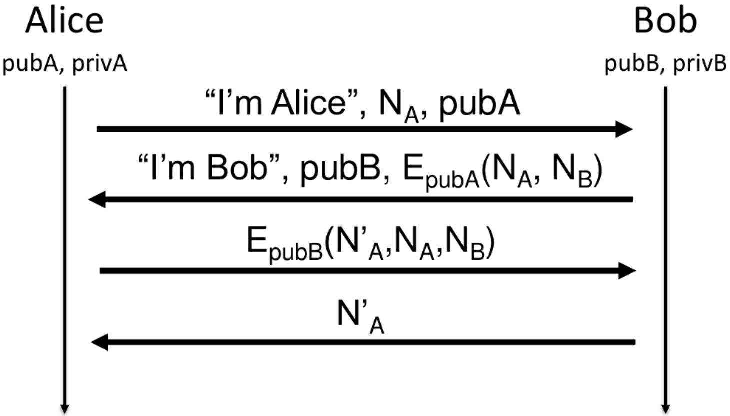


- Secure?

Asymmetric Key Authentication IV

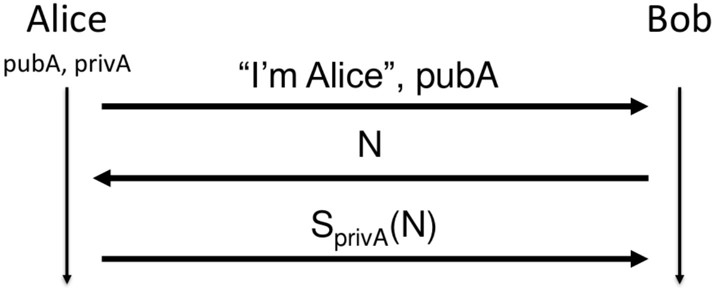
- Secure?
- YES!
- But don't forget the mutual authentication

Asymmetric Key Authentication V



Signature Authentication I

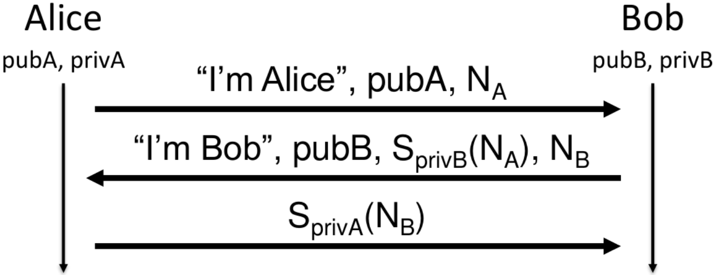
Signature authentication:



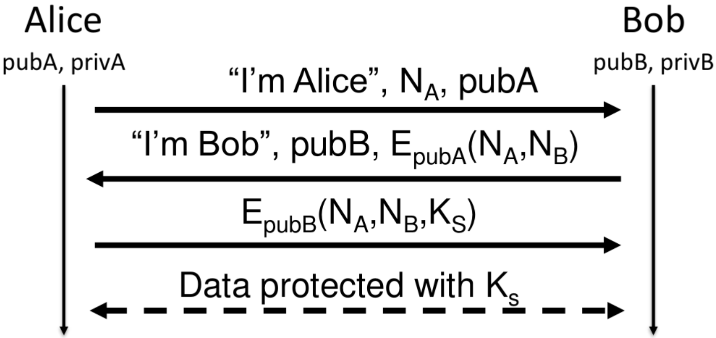
Signature Authentication II

- Secure?
- No
- Trudy can get Alice to sign anything!
- Same as previous, use different keys for different purposes

Signature Authentication III



Session key Authentication I



Diffie-Hellman I

- Used to authenticate
- DH does not guarantee PFS
- Vulnerable to MITM attacks (as seen in topic 2)
- Security -> Discrete Logarithm Problem

Diffie-Hellman II

Alice

Known	Unknown
$p = 23$	
$g = 5$	
$a = 6$	b
$A = 5^a \bmod 23$	
$A = 5^6 \bmod 23 = 8$	
$B = 19$	
$s = B^a \bmod 23$	
$s = 19^6 \bmod 23 = 2$	

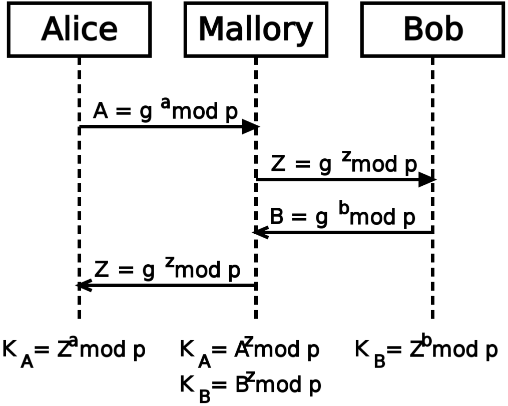
Bob

Known	Unknown
$p = 23$	
$g = 5$	
$b = 15$	a
$B = 5^b \bmod 23$	
$B = 5^{15} \bmod 23 = 19$	
$A = 8$	
$s = A^b \bmod 23$	
$s = 8^{15} \bmod 23 = 2$	

Eve

Known	Unknown
$p = 23$	
$g = 5$	
	a, b
$A = 8$, $B = 19$	
	s

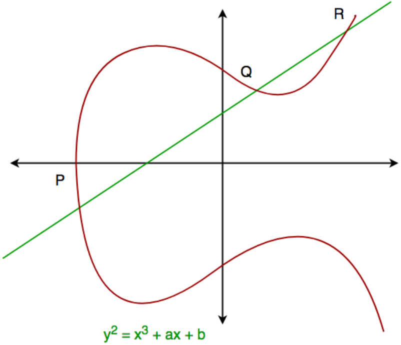
Diffie-Hellman III



Diffie-Hellman IV

- DH can be improved:
    - Using ephimeral keys -> guarantee PFS
    - Using ECC -> Shorter keys

Diffie-Hellman V



This approach uses six tuple {P, a, b, G, n, h}

P = Field that the curve is define over

G = Generator point

a, b = Values define the curve

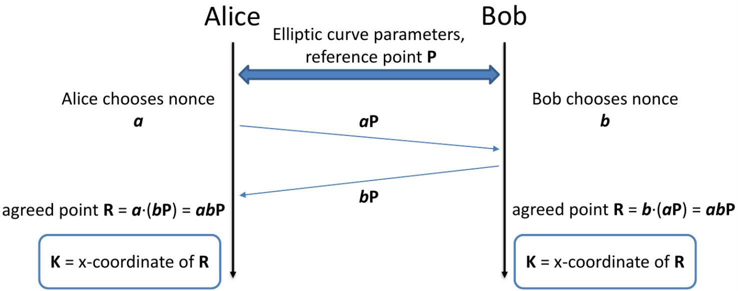
h = Co- factor

n = Prime order of G

Diffie-Hellman VI

ECDH:

- An attacker need a or b (points of the curve)
- Computationally unfeasible given only aP and bP



Diffie-Hellman VII

- With the ECC implementation with DH we can make this keys ephemeral
- Ephemeral mean that keys won't be reused
- Guaranteed PFS
- Elliptic Curve Diffie-Hellman Ephemeral

Authentication types

- We can divide the secrets that someone can know in 3:
  - Something you know (Passwords)
  - Something you have (Smart phone, Smart card)
  - Something you are (fingerprint, iris scan)
- 2 factor or multifactor authentication requires 2 or more methods
- We will consider a client/server paradigm for authentication

Something you know I

- Mainly used passwords!
- Passwords most of the time are not secure because of predictability of people
- But they are free, easy to use and easy to restore if lost
- Secure for standalone system, but insecure for networked system
- We can't send passwords in plain nor encrypted in a non-trustable network

Something you know II

- Passwords are in plain!!!

Something you know III

What we have:

- We know our password
- Server has to know our password or something related to the password

What we need to achieve this:

- Hash function (sending the has is better than sending the password)

Something you know IV

- The problem:
  - If we only hash an attacker can make a replay attack
  - Hashes are not 100% secure, rainbow tables, collisions...
- To ensure "freshness" we can add a nonce to the password or a **salt**
- This salt will be public and stored with the password hash
- If we want to improve the security (but reducing performance) we can hash many times
- Now the Server will store $H = Hash(password + salt)$

Something you know V

- However, passwords has other problems:
  - Short passwords (less than 10 characters)
  - Use of few symbols (lowercase, uppercase, numbers...)
  - Use of known things (pet's name, birth year...)
- These bad practices can be exploited by malicious actors
- Dictionary attacks

Something you know VI

- Password cracking tools:
  - John the ripper (we will see in lab)
  - HashCat
  - Cain & Abel
- Admins should use these tools to test for weak passwords

Something you know VII

Crypto keys

- Minimum key length is usually 128 bits
- Then 2^{128} different keys
- Choose key at random…
- …then attacker must try about 2^{127} keys

Passwords

- Passwords are usually 8 characters
- Assuming 256 different characters, then $256^8 = 2^{64}$ different passwords
- However, **less than the 256 ASCII characters are used**:
  - Often just [a-z],[A-Z],[0-9]: 62 characters
  - Then $62^8 \approx 2^{47} \ll 2^{64}$ passwords
- **It's only 47 bits!**
- Moreover, **users do not select passwords at random**

Something you have

- Relies on something you have to authenticate
- For example:
    - Google's login
    - OTP (One Time Passwords) & TOTP (Time-based OTP)
    - SMS
    - Hardware

Something you are I

- You are your own key
- For example:
  - Fingerprint
  - Facial Recognision
  - Iris scan

Something you are II

- They are more secure than others (nothing to remember)
- Not shareable
- They are fairly unique
- Ideally:
  - Universal: Applies to everyone
  - Distinguishing: Distinguish with certainty
  - Permanent: Characteristic that never changes
  - Collectable: Easy to collect data

Something you are III

- Their cost is higher than the others
- Difficult in some devices to use them

Biometric	Accuracy	Cost	Devices required
Fingerprint	Medium	Low	Scanner
Hand geometry	Low	Low	Scanner
Face recognition	Low	Medium-High	Camera
Iris scan	High	High	Camera
Voice recognition	Medium	Medium	Microphone
ADN	High	High	Test equipment

What is Kerberos I

- The three headed dog guardian of the gate of the kingdom of Hades (god of death) from greek mythology

What is Kerberos II

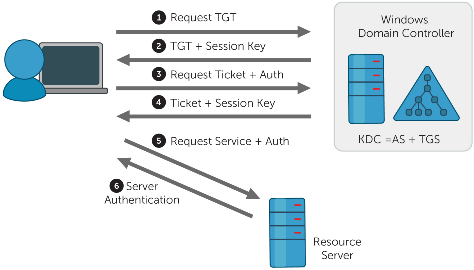
- Computer-network security protocol to authenticate 2 hosts across an untrusted network
- Written in C
- Developed in late 80s by MIT
- Default protocol Windows since Windows XP
- Used in Active Directory, NFS, and Samba
- Also an alternative authentication system to SSH, POP, and SMTP

How it works I

- Trusted Third-Party (TTP)
- Kerberos uses **tickets** to authenticate!
- Agents in kerberos authentication:
    - Key Distribution Center (KDC): Issues, distributes and authenticates tickets
        - Ticket Granting Server (TGS): Issues and distributes the tickets
        - Authentication Server (AS): Authenticates the tickets
    - Ticket Granting Ticket (TGT): Ticket used to obtain STs
    - Service Ticket (ST): Ticket to get access to a service

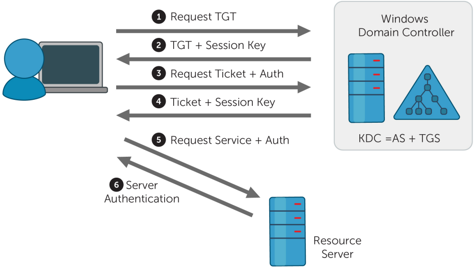
How it works II

1. Client asks for TGT to KDC with userID and a secret key derived from password.



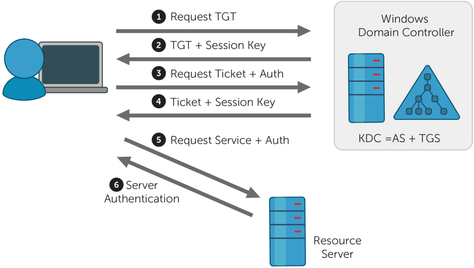
How it works III

2. KDC answers with TGT and session key. Communication encrypted with user's pasword.



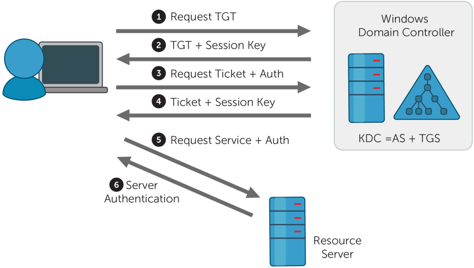
How it works IV

3. Client asks for service ticket sending the target server, TGT and an authenticator derived session key.



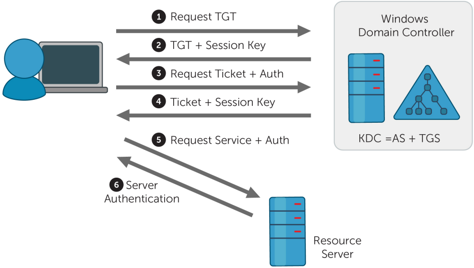
How it works V

4. KDC answers with the ticket for the service and session key.



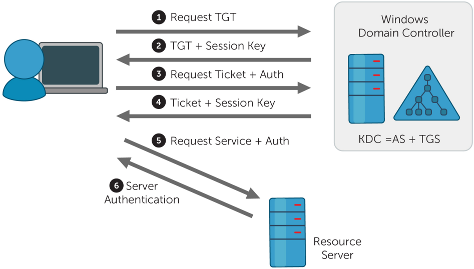
How it works VI

5. Client asks resources to server with the ticket.



How it works VII

6. Server checks the validity of the ticket decrypting with server's password. If it is valid, success in authentication!



- Optionally the server can check the validity of the ticket to the KDC.

Security I

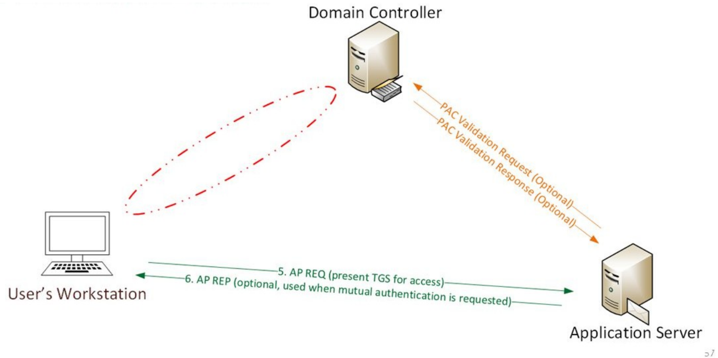
- Do you think it is secure?

Security II

- Do you think it is secure?
- Yes and no
- Some example of attacks:
  - Pass-the-Ticket: Steal a ticket and get access to the service
  - Silver Ticket: stealing the password or password hash of target machine to forge STs
  - Golden Ticket: Steal the password or password hash that encrypts TGT (for instance, in AD krbtgt) to forge custom TGTs
  - Kerberoasting: Harvest ST for services and crack the credentials offline.

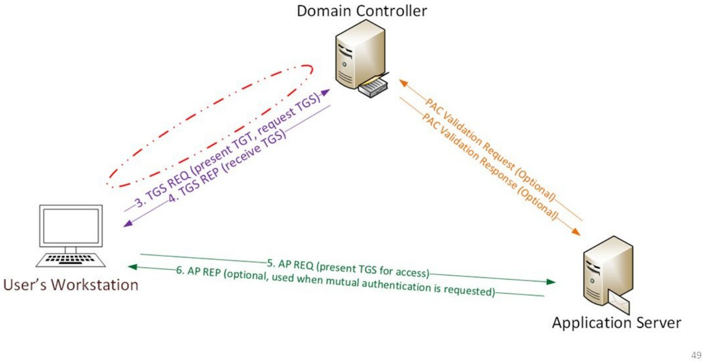
Security III

- Silver Ticket:



Security IV

- Golden Ticket:



The END!