

# Access control models

# Access types

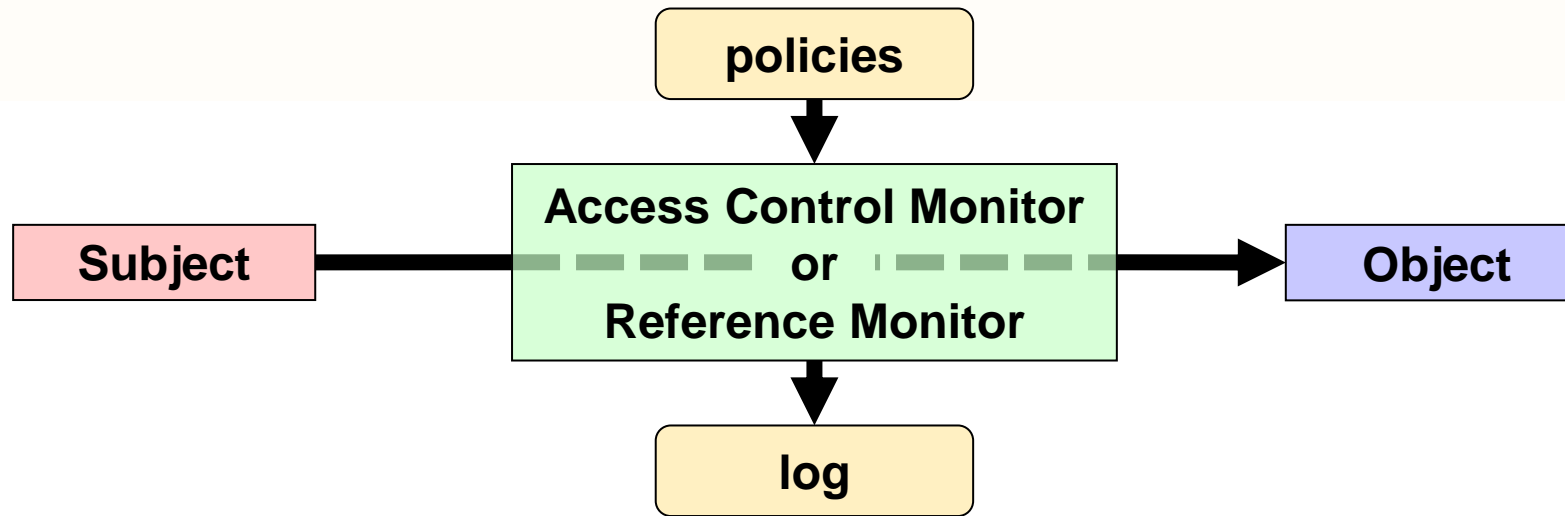
## ▷ Physical access

- ♦ Physical contact between a subject and the object of interest
  - Facility, room, network, computer, storage device, authentication token, etc.
- ♦ **Out of scope of this course ...**

## ▷ Informatic or electronic access

- ♦ Information-oriented contact between a subject and the object of interest
  - Contact through request-response dialogs
- ♦ Contact is mediated by
  - Computers and networks
  - Operating systems, applications, middleware, devices, etc.

# Access control



## ▷ Definition

- ♦ The policies and mechanisms that mediate the access of a subject to an object

## ▷ Normal requirements

- ♦ Authentication
  - With some Level of Assurance (LoA)
- ♦ Authorization
- ♦ Accountability → logging

} AAA

# Access control

## ▷ Subjects and objects

- ♦ Both digital entities
- ♦ Subjects can be something exhibiting activity :
  - Processes
  - Computers
  - Networks
- ♦ Objects can be the target of an action :
  - Stored data
  - CPU time
  - Memory
  - Processes
  - Computers
  - Network

## ▷ An entity can be both subject and object

# Least privilege principle

Every program and every user of the system should operate using the least set of privileges necessary to complete the job

J. H. Saltzer, M. D. Schroeder,

The protection of information in computer systems, Proc. of the IEEE, 63(9) 1975

- ▷ Privilege:
  - ♦ Authorization to perform a given task
  - ♦ Similar to access control clearance
- ▷ Each subject should have, at any given time, the exact privileges required to the assigned tasks
  - ♦ Less privileges than the required create unsurpassable barriers
  - ♦ More privileges than the required create vulnerabilities
    - Damage resulting from accidents or errors
    - Potential interactions among privileged programs
    - Misuse of a privileges
    - Unwanted information flows
      - "need-to-know" military restrictions

# Access control models

	O1	O2	...	O <sub>m-1</sub>	O <sub>m</sub>
S1		Access rights			
S2					
...					
S <sub>n-1</sub>					
S <sub>n</sub>					

## ▷ Access control matrix

- ♦ Matrix with all access rights for subjects relatively to objects
- ♦ Represents a conceptual model of the organization

# Access control models

	O1	O2	...	O <sub>m-1</sub>	O <sub>m</sub>
S1		Access rights			
S2					
...					
S <sub>n-1</sub>					
S <sub>n</sub>					

## ▷ ACL-based mechanisms

- ♦ **ACL: Access Control List (matrix column)**
  - List of access rights for specific subjects
  - Access rights can be positive or negative
  - Default subjects may often be used
- ♦ **Usually ACLs are stored along with objects**
  - e.g. for file system objects.
- ♦ **Rights are then mapped to specific actions**
  - Same right may map to different actions on different contexts

# Access control models

	O1	O2	...	O <sub>m-1</sub>	O <sub>m</sub>
S1		Access rights			
S2					
...					
S <sub>n-1</sub>					
S <sub>n</sub>					

## ▷ Capability-based mechanisms

- ♦ **Capability: unforgeable authorization token (matrix row)**
  - Contains object references and access rights
- ♦ **Access granting**
  - Transmission of capabilities between subjects
- ♦ **Usually capabilities are kept by subjects**
  - e.g. OAuth 2.0 access tokens



# Access control kinds:

## MAC and DAC

### ▷ Mandatory access control (MAC)

- ♦ Access control policy statically implemented by the access control monitor
- ♦ Access control rights cannot be tailored by subjects or object owners

### ▷ Discretionary access control (DAC)

- ♦ Some subjects can update rights granted or denied to other subjects for a given object
  - Usually this is granted to object owners and system administrators

# Access control kinds:

## Role-Based Access Control (RBAC)

D.F. Ferraiolo and D.R. Kuhn, "Role Based Access Control", 15th National Computer Security Conference, Baltimore, October 1992

### ▷ Not DAC or MAC

- ♦ Roles are dynamically assigned to subjects
  - For access control it matters the role played by the subject and not the subject's identity

### ▷ Access control binds roles to (meaningful) operations

- ♦ Operations are complex, meaningful system transactions
  - Not the ordinary, low-level read/write/execute actions on individual objects
- ♦ Operations can involve many individual lower-level objects

# Access control kinds:

## RBAC rules (1/2)

### ▷ Role assignment:

- ♦ All subject activity on the system is conducted through transactions
  - And transactions are allowed to specific roles
  - Thus all active subjects are required to have some active role
- ♦ A subject can execute a transaction iff
  - it has selected
- ♦ or
  - been assigned
  - a role which can use the transaction

# Access control kinds:

## RBAC rules (2/2)

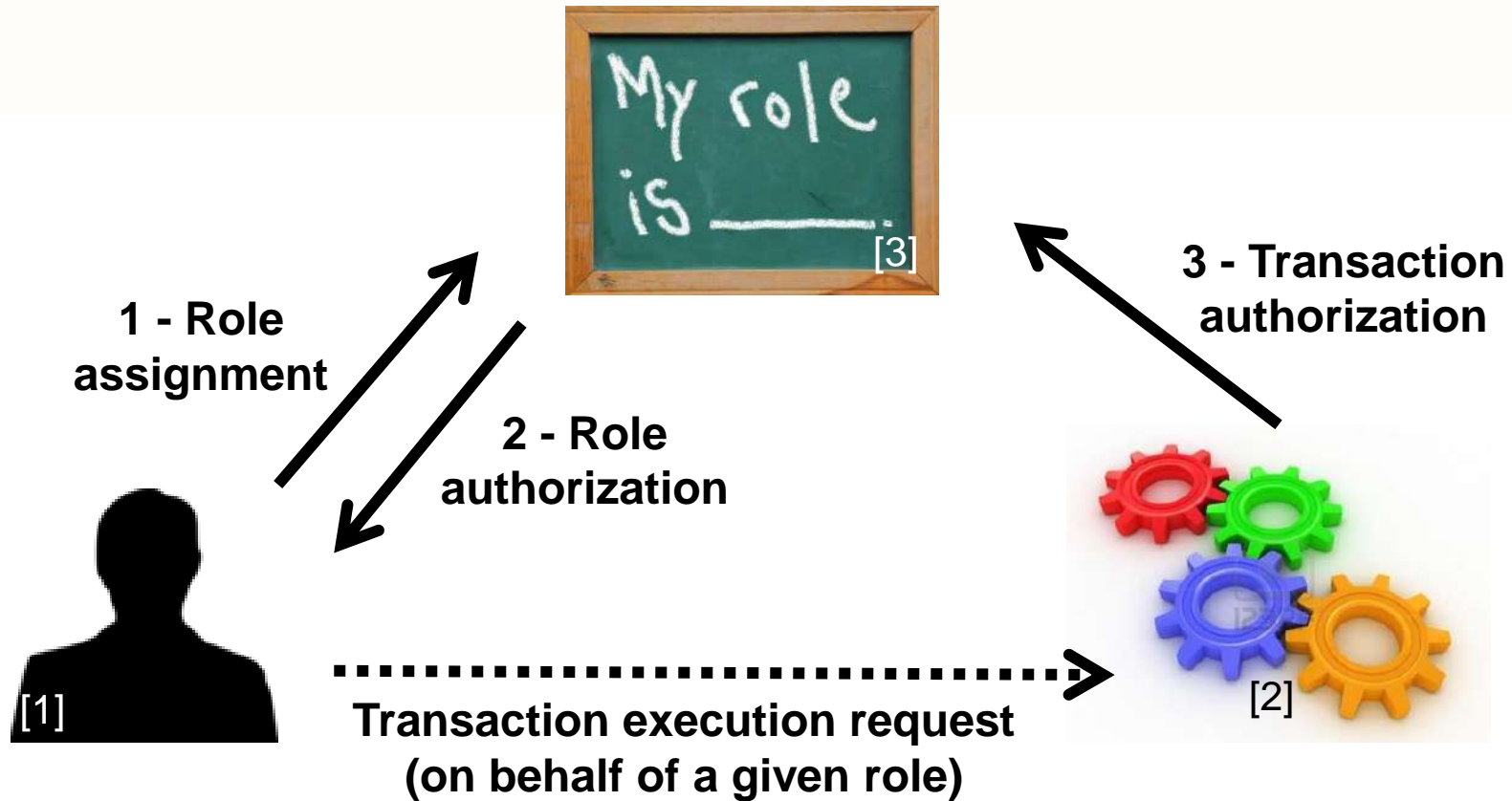
### ▷ Role authorization:

- ♦ A subject's active role must be authorized for the subject

### ▷ Transaction authorization:

- ♦ A subject can execute a transaction **iff**
  - the transaction is authorized through the subject's role memberships
  - and
  - there are no other constraints that may be applied across subjects, roles, and permissions

# RBAC rules



[1] From <http://www.clker.com/clipart-24011.html>

[2] From [http://www.123rf.com/photo\\_12115593\\_three-dimensional-colored-toothed-wheels.html](http://www.123rf.com/photo_12115593_three-dimensional-colored-toothed-wheels.html)

[3] From <http://www1.yorksolutions.net/Portals/115255/images/MyRoleIs.jpg>

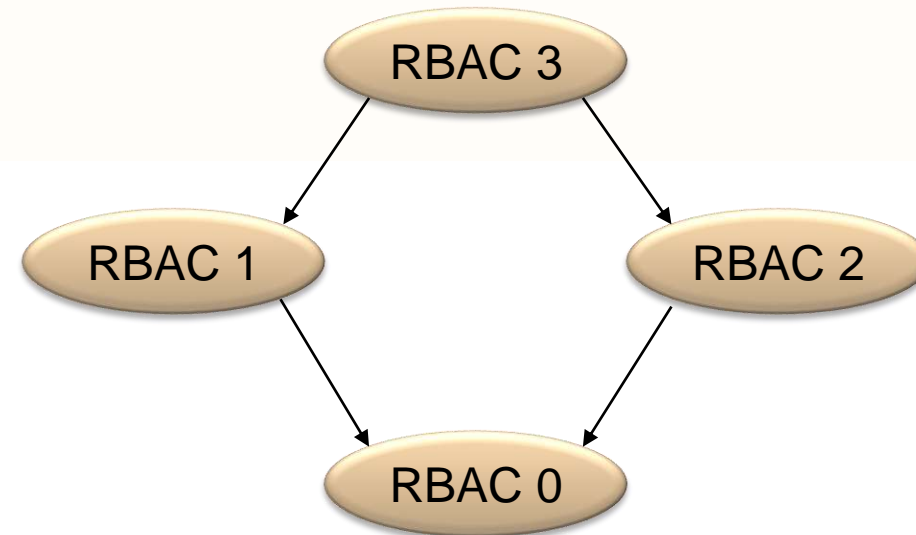
# RBAC:

## Roles vs. groups

- ▷ Roles are a collection of permissions
  - ♦ The permissions are granted to the subjects that, at a given instant, play the role
  - ♦ A subject can only play a role at a given time
  
- ▷ Groups are a collection of users
  - ♦ And permissions can be granted both to users and groups
  - ♦ A subject can belong to many groups at a given time
  
- ▷ The session concept
  - ♦ Role assignment is similar to a session activation
  - ♦ Group membership is ordinarily a static attribute

# RBAC variants

- ▷ RBAC 0
  - ◆ No role hierarchies
  - ◆ No role constraints
- ▷ RBAC 1
  - ◆ RBAC 0 w/ role hierarchies (privilege inheritance)
- ▷ RBAC 2
  - ◆ RBAC 0 w/ role constraints (separation of duties)
- ▷ RBAC 3
  - ◆ RBAC 1 + RBAC 2



# NIST RBAC model

## ▷ Flat RBAC

- ♦ Simple RBAC model w/ user-role review
- ♦ Role provides specific permissions for the user

## ▷ Hierarchical RBAC

- ♦ Flat RBAC w/ role hierarchies (DAG or tree)
- ♦ General and restricted hierarchies, where Roles gain additional permissions from other roles

## ▷ Constraint RBAC

- ♦ RBAC w/ role constraints for separation of duty
- ♦ Static: Conflicting Roles cannot be assigned
- ♦ Dynamic: Subject cannot activate conflicting Roles within session

## ▷ Symmetric RBAC

- ♦ RBAC w/ organization wide permission-role review
- ♦ Allows review of a subject roles to prevent bloat



# Access control kinds:

## Context-Based Access Control (CBAC)

- ▷ Access rights have an historical context
  - ♦ The access rights cannot be determined without reasoning about past access operations
  - ♦ Example:
    - Stateful packet filter firewall
- ▷ Chinese Wall policy
  - ♦ Conflict groups
  - ♦ Access control policies need to address past accesses to objects in different members of conflict groups

D.F.C. Brewer and M.J. Nash, "The Chinese Wall Security Policy",  
IEEE Symposium on Security and Privacy, 1989

# Access control kinds:

## Attribute-Based Access Control (ABAC)

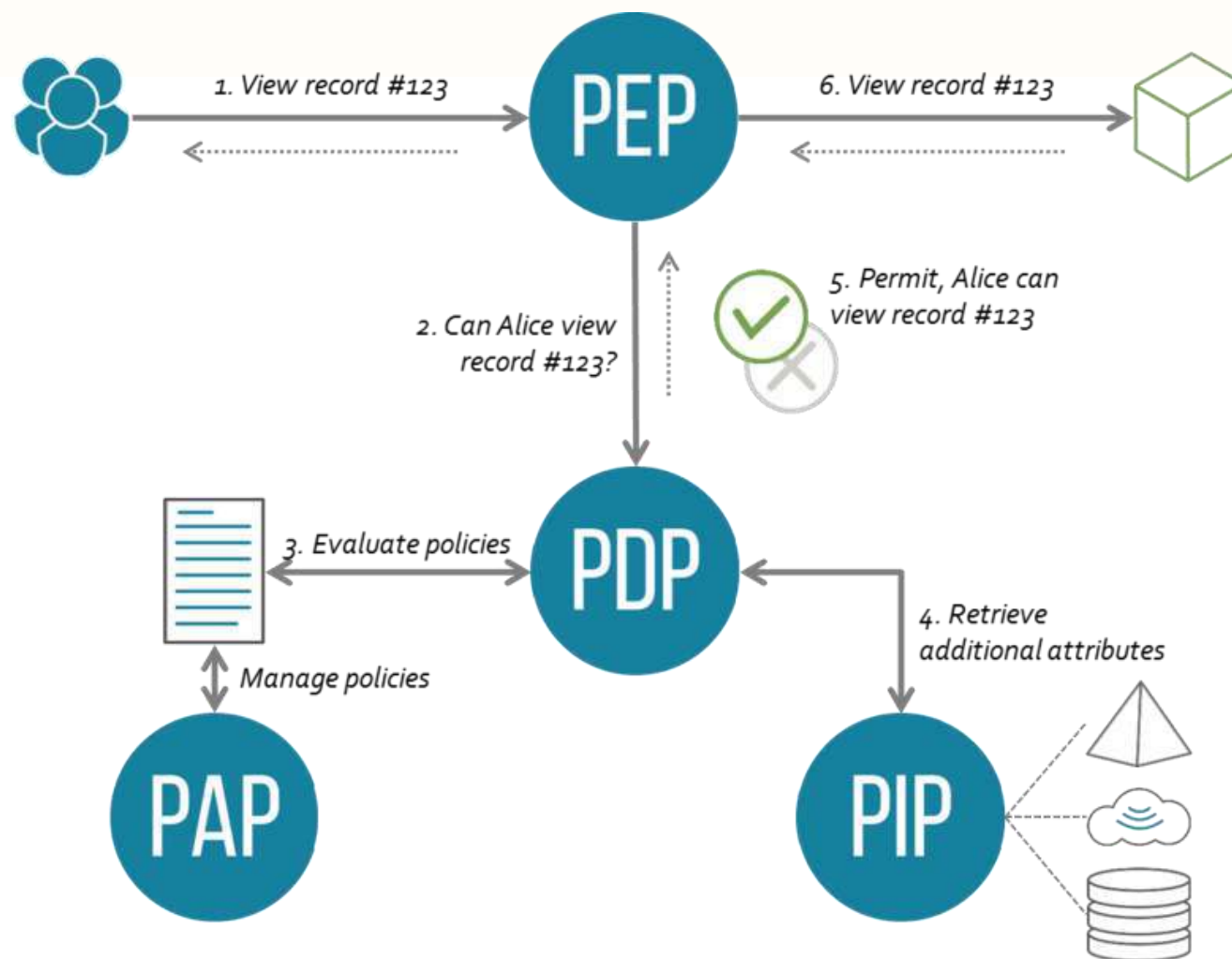
- ▷ Access control decisions are made based on attributes associated with relevant entities
  
- ▷ OASIS XACML architecture
  - ◆ Policy Administration Point (PAP)
    - Where policies are managed
  - ◆ Policy Decision Point (PDP)
    - Where authorization decisions are evaluated and issued
  - ◆ Policy Enforcement Point (PEP)
    - Where access requests to a resource are intercepted and confronted with PDP's decisions
  - ◆ Policy Information Point (PIP)
    - Provides external information to a PDP

# XACML:

## Access control with PEP and PDP

- ▷ A subject sends a request
  - ◆ Which is intercepted by the Policy Enforcement Point (PEP)
- ▷ The PEP sends the authorization request to the Policy Decision Point (PDP)
- ▷ The PDP evaluates the request against its policies and reaches a decision
  - ◆ Which is returned to the PEP
  - ◆ Policies are retrieved from a Policy Retrieval Point (PRP)
  - ◆ Useful attributes are fetched from Policy Information Points (PIP)
  - ◆ Policies are managed by the Policy Administration Point (PAP)

# XACML big picture



From <https://en.wikipedia.org/wiki/XACML>

# Break-the-glass access control model

- ▷ It may be required to overcome the established access limitations
  - ♦ e.g. in a life threatening situation
- ▷ The subject may be presented with a break-the-glass decision upon a deny
  - ♦ Can overcome the deny at their own responsibility
  - ♦ Logging is fundamental to prevent abuses
    - Subject may have to justify action, after using the elevated right

# Separation of duties

R.A. Botha, J.H.P. Eloff, "Separation of duties for access control enforcement in workflow environments", IBM Systems Journal, 2001

- ▷ Fundamental security requirement for fraud and error prevention
  - ◆ Dissemination of tasks and associated privileges for a specific business process among multiple subjects
  - ◆ Often implemented with RBAC
  
- ▷ Damage control
  - ◆ Segregation of duties helps reducing the potential damage from the actions of one person
  - ◆ Some duties should not be combined into one position

# Segregation of duties:

## ISACA (Inf. Systems Audit and Control Ass.) matrix guideline

Exhibit 2.9—Segregation of Duties Control Matrix													
	Control Group	Systems Analyst	Application Programmer	Help Desk and Support Manager	End User	Data Entry	Computer Operator	Database Administrator	Network Administrator	Systems Administrator	Security Administrator	Systems Programmer	Quality Assurance
Control Group		X	X	X		X	X	X	X	X		X	
Systems Analyst	X			X	X		X				X		X
Application Programmer	X			X	X	X	X	X	X	X	X	X	X
Help Desk and Support Manager	X	X	X		X	X		X	X	X		X	
End User		X	X	X			X	X	X			X	X
Data Entry	X		X	X			X	X	X	X	X	X	
Computer Operator	X	X	X		X	X		X	X	X	X	X	
Database Administrator	X		X	X	X	X	X		X	X		X	
Network Administrator	X		X	X	X	X	X	X					
System Administrator	X		X	X		X	X	X				X	
Security Administrator		X	X			X	X					X	
Systems Programmer	X		X	X	X	X	X	X		X	X		X
Quality Assurance		X	X		X							X	

X marks an incompatibility

X—Combination of these functions may create a potential control weakness.

# Segregation of duties:

## Declaração de Práticas de Certificação da EC do Cartão de Cidadão

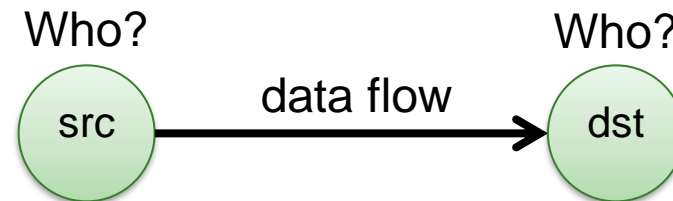
	Administração de Sistemas	Operação de Sistemas	Administração de Segurança	Auditoria de Sistemas	Custódia	Manutenção Sistemas de Suporte	Gestão
Administração de Sistemas			X	X	X	X	X
Operação de Sistemas			X	X	X	X	X
Administração de Segurança	X	X		X	X	X	X
Auditoria de Sistemas	X	X	X		X	X	X
Custódia	X	X	X	X		X	X
Manutenção Sistemas de Suporte	X	X	X	X	X		X
Gestão	X	X	X	X	X	X	

X marks an incompatibility



# Information flow models

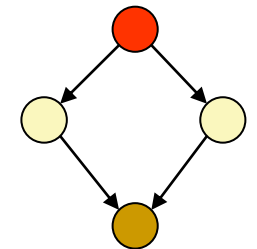
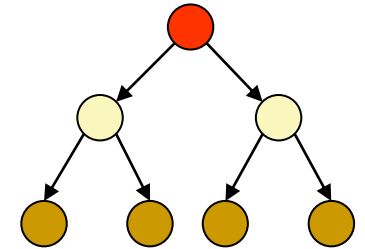
- ▶ Authorization is applied to data flows
  - ♦ Considering the data flow source and destination
  - ♦ Goal: avoid unwanted/dangerous information flows



- ▶ Src and Dst security-level attributes
  - ♦ Information flows should occur only between entities with given **security-level (SL)** attributes
  - ♦ Authorization is given based on the **SL** attributes

# Multilevel security

- ▷ Subjects (or roles) act on different security levels
  - ♦ Levels do not intersect themselves
  - ♦ Levels have some partial order
    - Hierarchy
    - Lattice
- ▷ Levels are used as attributes of subjects and objects
  - ♦ Subjects: security level clearance
  - ♦ Objects: security classification
- ▷ Information flows & security levels
  - ♦ Same security level → authorized
  - ♦ Different security levels → controlled
    - Authorized or denied on a “need to know” basis



# Multilevel security levels:

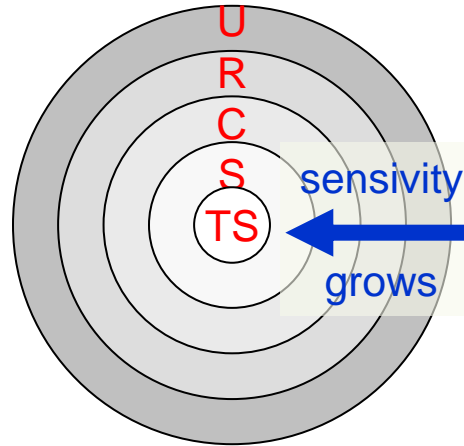
## Military / Intelligence organizations

### ▷ Typical levels

- ◆ Top secret
- ◆ Secret
- ◆ Confidential
- ◆ Restricted
- ◆ Unclassified

### ▷ Portugal ([NTE01](#), [NTE04](#))

- ◆ Muito Secreto
- ◆ Secreto
- ◆ Confidencial
- ◆ Reservado



### ▷ EU example

- ◆ EU TOP SECRET
- ◆ EU SECRET
- ◆ EU CONFIDENTIAL
- ◆ EU RESTRICTED
- ◆ EU COUNCIL / COMMISSION

### ▷ NATO example:

- ◆ COSMIC TOP SECRET (CTS)
- ◆ NATO SECRET (NS)
- ◆ NATO CONFIDENTIAL (NC)
- ◆ NATO RESTRICTED (NR)

# Multilevel security levels:

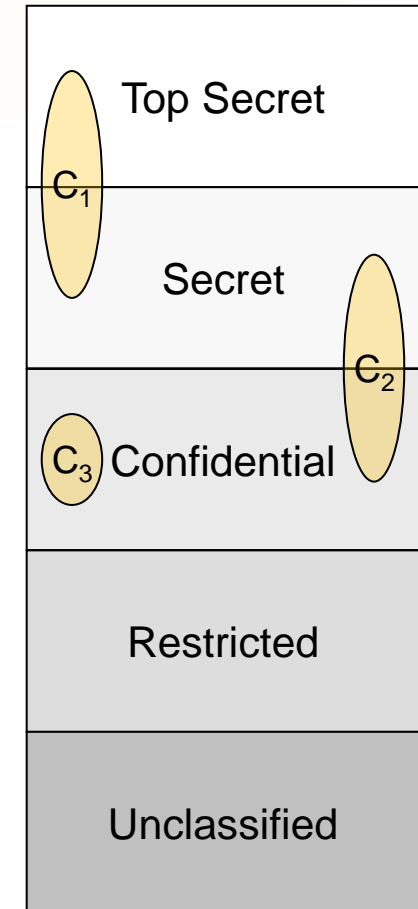
## Civil organizations

### ▷ Typical levels

- ◆ Restricted
- ◆ Proprietary
- ◆ Sensitive
- ◆ Public

# Security categories (or compartments)

- ▶ Self-contained information environments
  - ♦ May span several security levels
- ▶ Military environments
  - ♦ Military branches, military units
- ▶ Civil environments
  - ♦ Departments, organizational units
- ▶ An object can belong to different compartments and have a different security classification in each of them
  - (top-secret, crypto), (secret, weapon)



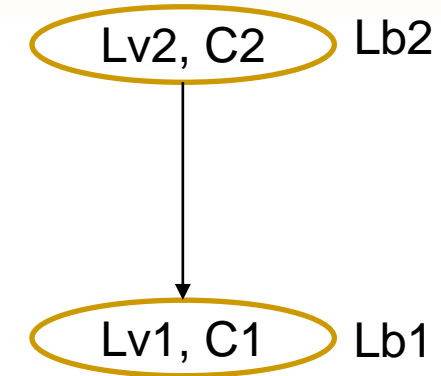
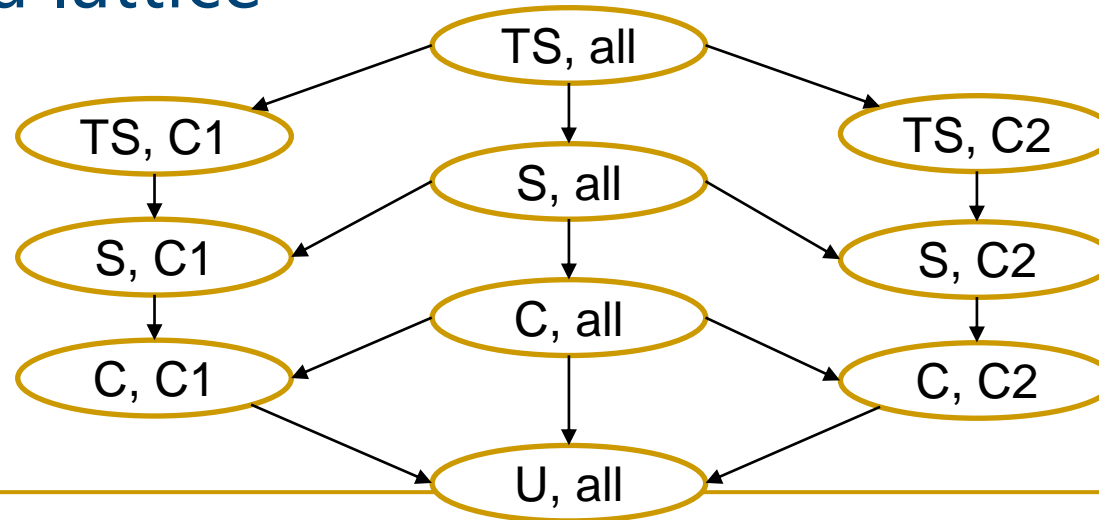
# Security labels

▷ Label = Category + Level

▷ Relative order between labels

$$Lb1 \leq Lb2 \Rightarrow C1 \subseteq C2 \wedge Lv1 \leq Lv2$$

▷ Labels form a lattice



# Bell-La Padula MLS Model

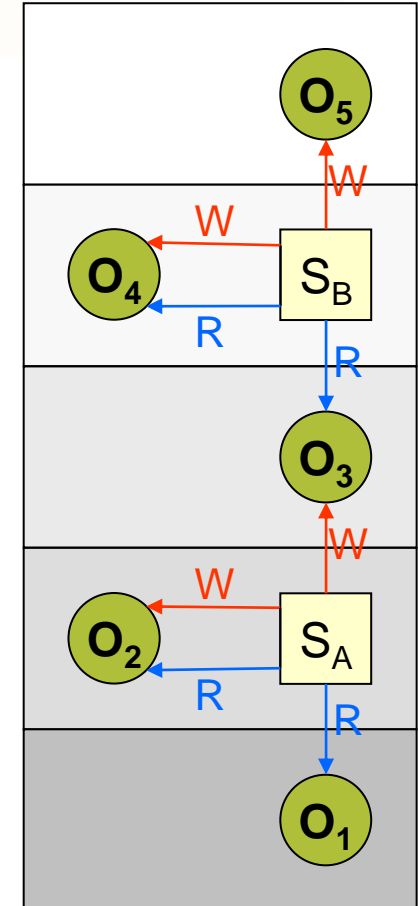
D. Elliott Bell, Leonard J. La Padula, "Secure Computer Systems: Mathematical Foundations", MITRE Technical Report 2547, Volume I, 1973

- ▷ Access control policy for controlling information flows
  - ◆ Addresses data confidentiality and access to classified information
  - ◆ Addresses disclosure of classified information
    - Object access control is not enough
    - One needs to restrict the flow of information from a source to authorized destinations
- ▷ Uses a state-transition model
  - ◆ In each state there are subjects, objects, an access matrix and the current access information
  - ◆ State transition rules
  - ◆ Security levels and clearances
    - Objects have a security labels
    - Subjects have security clearances
    - Both refer to security levels (e.g. CONFIDENTIAL)

# Bell-La Padula MLS Model:

## Secure state-transition model

- ▷ Simple security condition (no read up)
  - ♦  $S$  can read  $O$  iff  $L(S) \geq L(O)$
- ▷ \*-property (no write down)
  - ♦  $S$  can write  $O$  iff  $L(S) \leq L(O)$
  - ♦ aka confinement property
- ▷ Discretionary Security Property
  - ♦ DAC-based access control

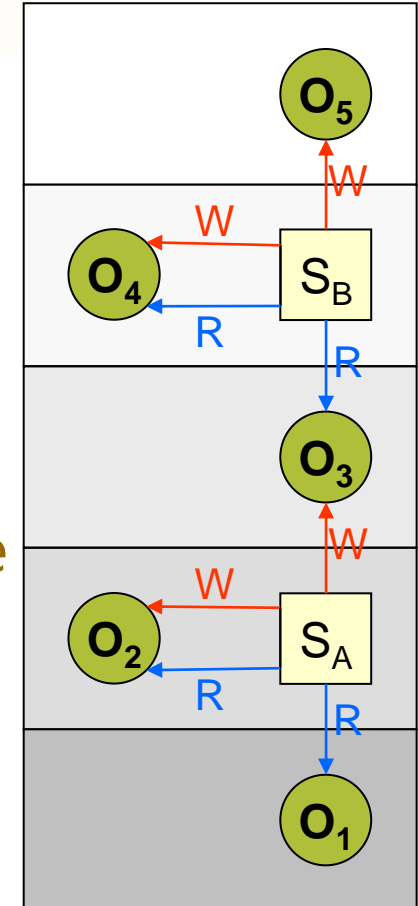




# Bell-La Padula MLS Model:

## Secure state-transition model

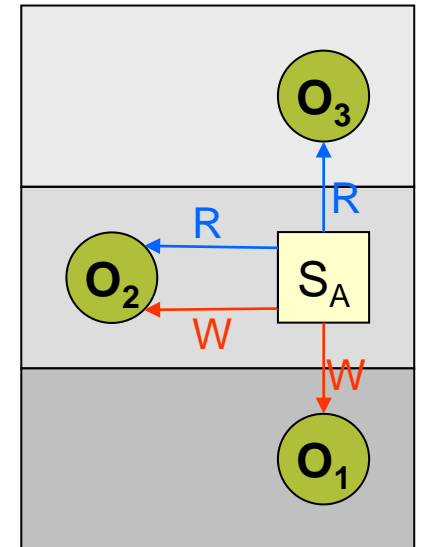
- ▷ Strong Star Property
  - ♦  $S$  can read  $O$  iff  $L(S) = L(O)$
- ▷ Tranquility Principle
  - ♦ Strong tranquility:  $S/O$  levels are static for the entire  $S/O$  lifetime
  - ♦ Weak tranquility:  $S/O$  levels may change if the security *spirit* of the system is not compromised
- ▷ Trusted Subjects
  - ♦  $S$  can write to lower levels



# Biba Integrity Model

K. J. Biba, "Integrity Considerations for Secure Computer Systems", MITRE Technical Report 3153, The Mitre Corporation, April 1977

- ▶ Access control policy for controlling information flows
  - ♦ For enforcing data integrity control
  - ♦ Uses integrity levels, not security levels
- ▶ Similar to Bell-La Padula, with inverse rules
  - ♦ Simple Integrity Property (no read down)
    - $S$  can read  $O$  iff  $I(S) \leq I(O)$
  - ♦ Integrity \*-Property (no write up)
    - $S$  can write  $O$  iff  $I(S) \geq I(O)$



# Biba Integrity Model

K. J. Biba, "Integrity Considerations for Secure Computer Systems", MITRE Technical Report 3153, The Mitre Corporation, April 1977

## ▷ Access control policy for controlling information flows

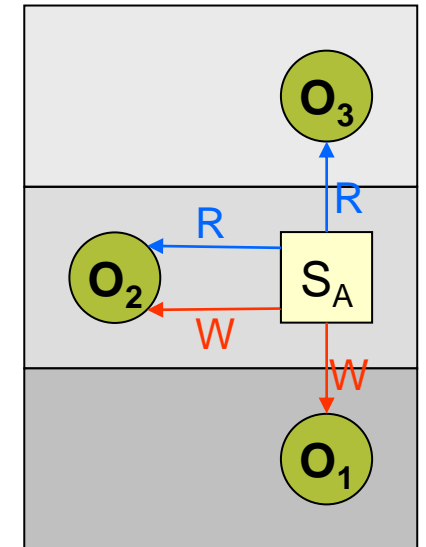
- ♦ For enforcing data integrity control
- ♦ Uses integrity levels, not security levels
- ♦ Subjects cannot corrupt objects at higher levels

## ▷ Similar to Bell-La Padula, with inverse rules

- ♦ Simple Integrity Property (no read down)
  - S can read O iff  $I(S) \leq I(O)$
- ♦ Integrity \*-Property (no write up)
  - S can write O iff  $I(S) \geq I(O)$

## ▷ Invocation Property

- ♦ S cannot request higher access



# Windows mandatory integrity control

- ▷ Allows mandatory (priority and critical) access control enforcement prior to evaluate DACLs
  - ◆ If access is denied, DACLs are not evaluated
  - ◆ If access is allowed, DACLs are evaluated
  
- ▷ Integrity labels
  - ◆ Untrusted
  - ◆ Low (or AppContainer)
  - ◆ Medium (default)
  - ◆ Medium Plus
  - ◆ High
  - ◆ System
  - ◆ Protected Process

DACL: discretionary access control list

<https://learn.microsoft.com/en-us/windows/win32/secauthz/mandatory-integrity-control>

# Windows mandatory integrity control

## ▷ Users

- ♦ **Medium**: standard users
- ♦ **High**: elevated users

## ▷ Process integrity level

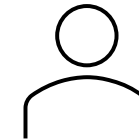
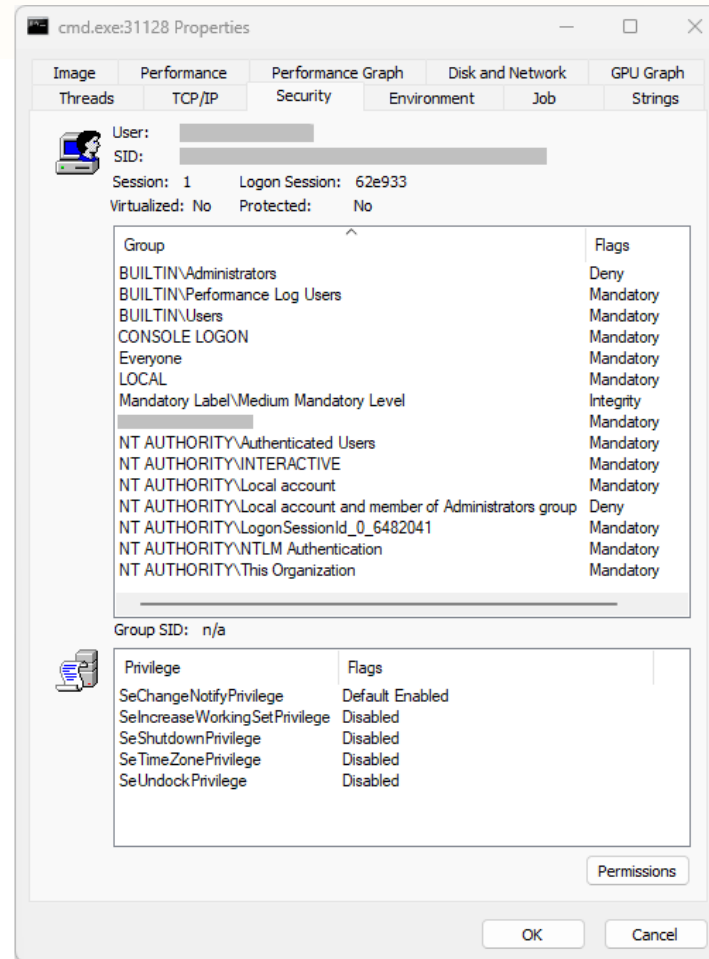
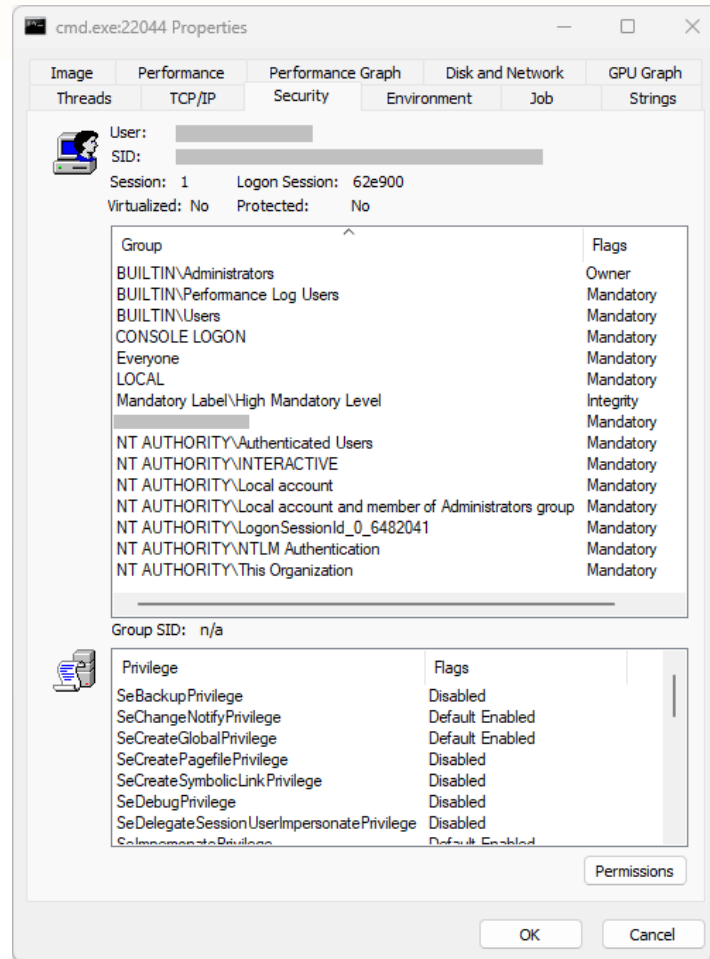
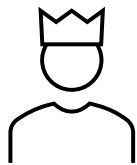
- ♦ The minimum associated to the owner and the executable file
- ♦ User processes usually are **Medium** or **High**
  - Except if executing **Low**-labeled executables
- ♦ Service processes: **High**

# Windows mandatory integrity control

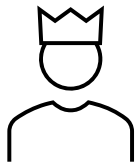
## ▷ Securable objects mandatory label

- ◆ NO\_WRITE\_UP (default)
- ◆ NO\_READ\_UP
- ◆ NO\_EXECUTE\_UP

# Windows mandatory integrity control

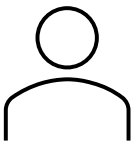


# Windows mandatory integrity control



```
D:\>icacls bar.txt /setintegritylevel(oi)(c) High
processed file: bar.txt
Successfully processed 1 files; Failed processing 0 files

D:\>icacls bar.txt
bar.txt BUILTIN\Administrators:(I)(F)
        NT AUTHORITY\SYSTEM:(I)(F)
        NT AUTHORITY\Authenticated Users:(I)(M)
        BUILTIN\Users:(I)(RX)
        Mandatory Label\High Mandatory Level:(NW)
```



```
D:\>echo "foo" > bar.txt

D:\>icacls bar.txt
bar.txt BUILTIN\Administrators:(I)(F)
        NT AUTHORITY\SYSTEM:(I)(F)
        NT AUTHORITY\Authenticated Users:(I)(M)
        BUILTIN\Users:(I)(RX)
```

```
D:\>echo 1234 > bar.txt
Access is denied.

D:\>del bar.txt
D:\bar.txt
Access is denied.
```

Time



# Clark-Wilson Integrity Model

D. D. Clark, D. R. Wilson, "A Comparison of Commercial and Military Computer Security Policies", IEEE Symposium on Security and Privacy, 1987

- ▷ Addresses information integrity control
  - ◆ Uses the notion of transactional data transformations
  - ◆ Separation of duty: transaction certifiers  $\neq$  implementers
- ▷ Terminology
  - ◆ Data items
    - Constrained Data Item (**CDI**)
      - Can only be manipulated by TPs
    - Unconstrained Data Item (**UDI**)
  - ◆ Integrity policy procedures
    - Integrity Verification Procedure (**IVP**)
      - Ensures that all CDIs conform with the integrity specification
    - Transformation Procedure (**TP**)
      - Well-formed transaction
        - Take as input a CDI or a UDI and produce a CDI
      - Must guarantee (via certification) that transforms all possible UDI values to "safe" CDI values

# Clark-Wilson Integrity Model: Certification & Enforcement

## ▷ Integrity assurance

- ◆ Certification

- Relatively to the integrity policy

- ◆ Enforcement

## ▷ Two sets of rules

- ◆ Certification Rules (C)

- ◆ Enforcement Rules (E)

# Clark-Wilson Integrity Model:

## Certification & Enforcement rules

- ▷ Basic rules
  - C1:** when an IVP is executed, it must ensure that all CDIs are valid
  - C2:** for some associated set of CDIs, a TP must transform those CDIs from one valid state to another
  - E1:** the system must maintain a list of certified relations and ensure only TPs certified to run on a CDI change that CDI
- ▷ Separation of duty (external consistency)
  - E2:** the system must associate a user with each TP and set of CDIs. The TP may access CDIs on behalf of the user if authorized
  - C3:** allowed user-TP-CDI relations must meet "separation of duty" requirements
- ▷ Identification gathering
  - E3:** the system must authenticate every user attempting a TP (on each attempt)
- ▷ Audit trail
  - C4:** all TPs must append to a log enough information to reconstruct operations
- ▷ UDI processing
  - C5:** a TP taking a UDI as input may only perform valid transactions for all possible values of the UDI. The TP will either accept (convert to CDI) or reject the UDI
- ▷ Certification constraints
  - E4:** only the certifier of a TP may change the associated list of entities

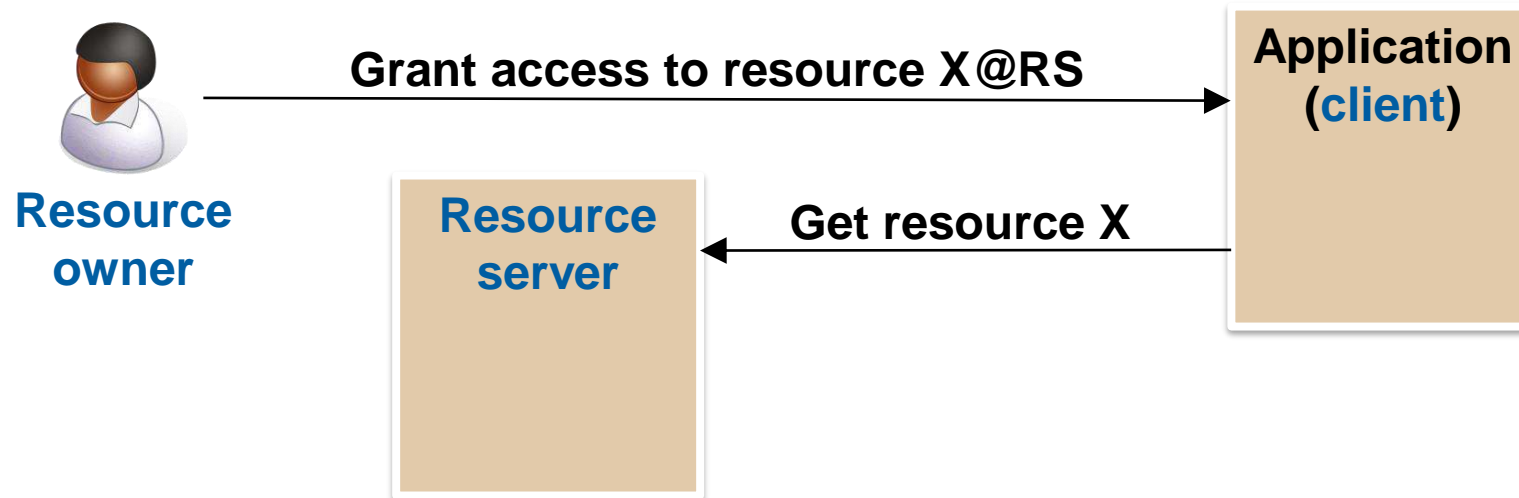
# OAuth 2.0

## authorization framework



# Goal

- ▶ Allow an application to access user resources maintained by a service/server



- ▶ Full reference at <https://oauth.net/2/>

# Roles (RFC 6749)

## ▷ Resource owner

- ♦ An entity capable of **granting access** to a **protected resource**
- ♦ **End-user**: a resource owner that is a person

## ▷ Resource server

- ♦ The server hosting protected resources
- ♦ Capable of accepting and responding to protected resource requests using **access tokens**

# Roles (RFC 6749)

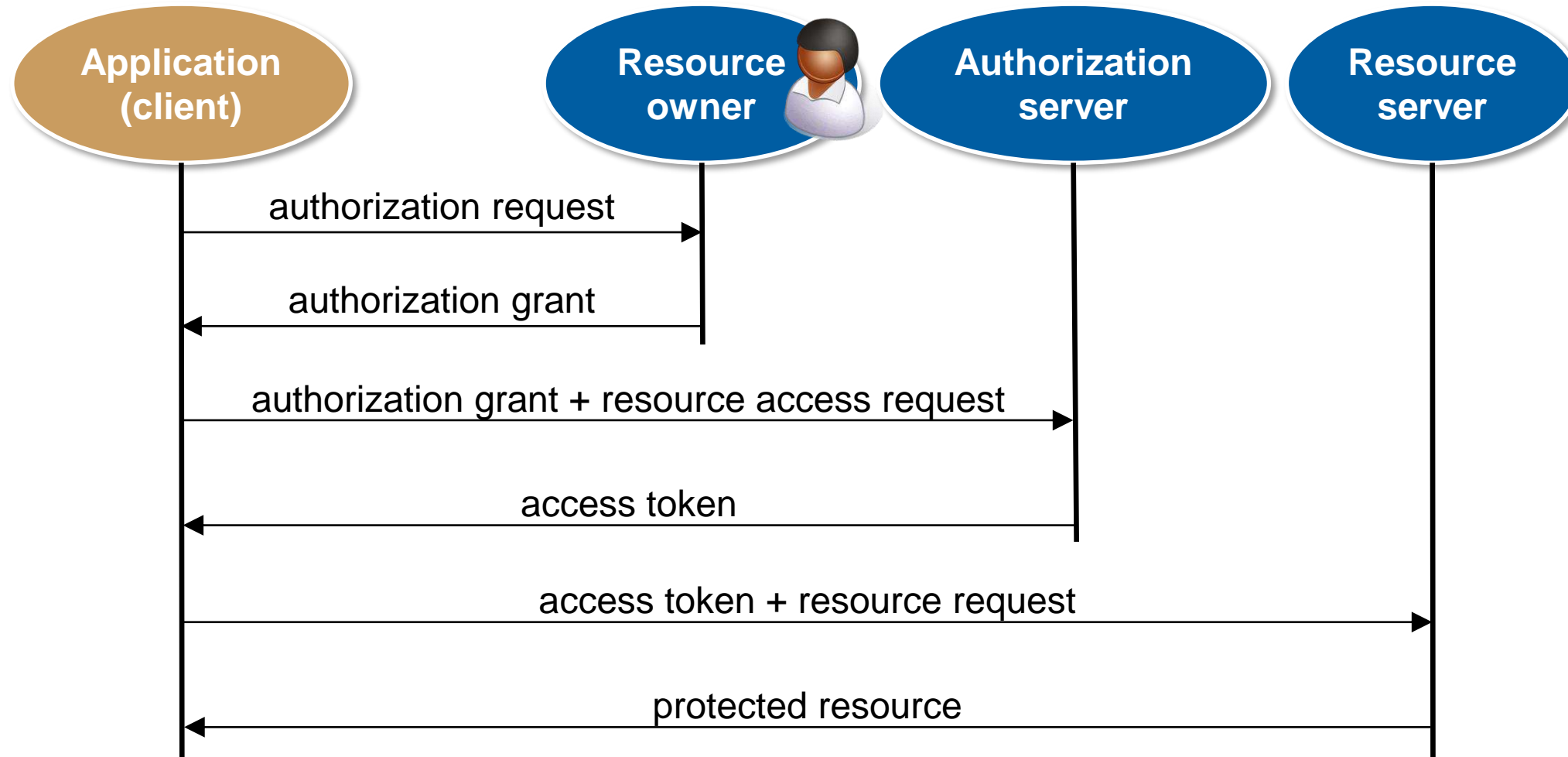
## ▷ Client

- ♦ An **application** making requests for protected resources on behalf of the resource owner and with its authorization

## ▷ Authorization server (aka OAuth server or provider)

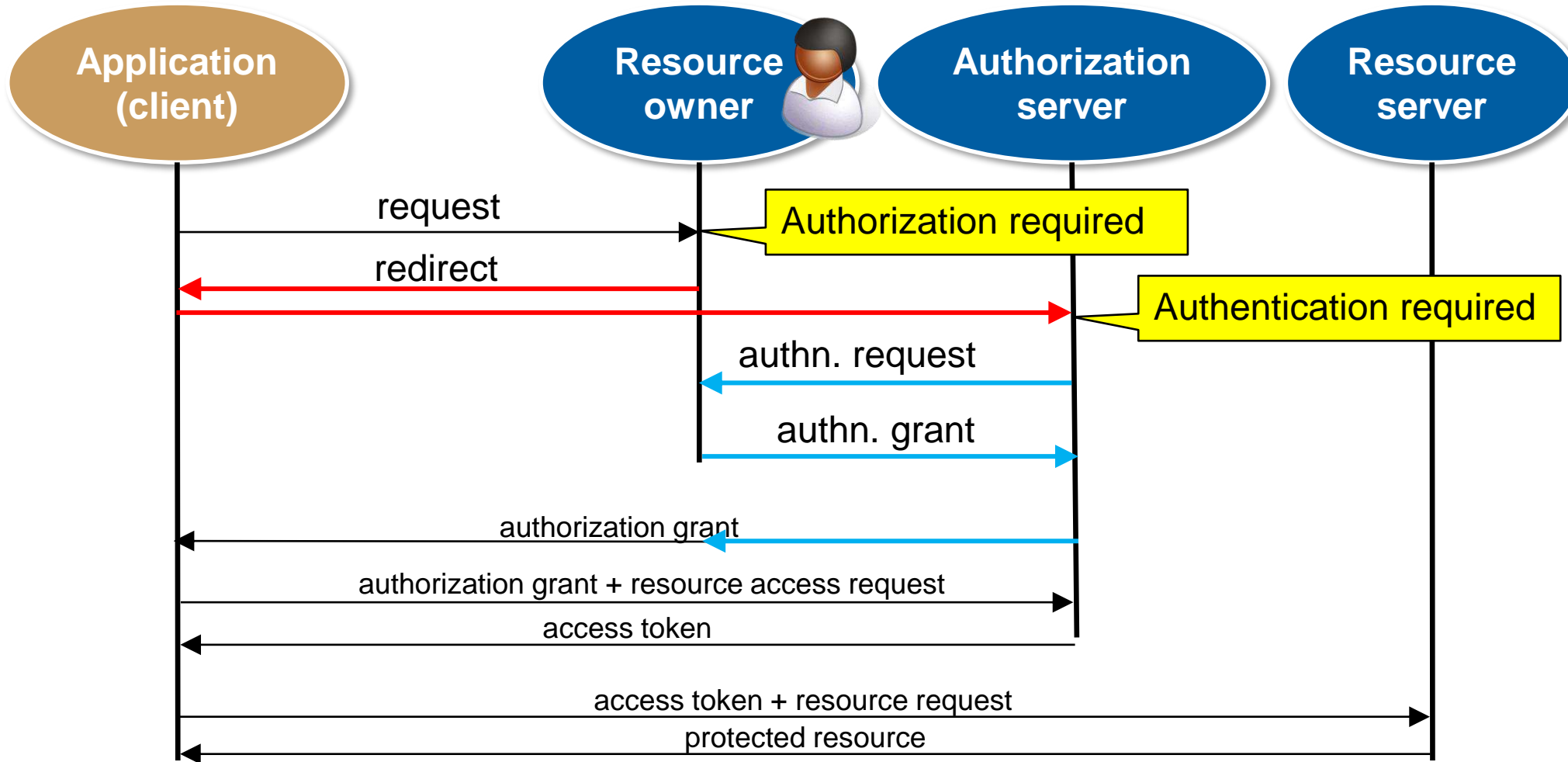
- ♦ The server issuing **access tokens** to the client after successfully **authenticating** the resource owner and obtaining its **authorization** for the client to access one of its resources

# Abstract protocol flow (RFC 6749)





# Common protocol flow



# Communication endpoints: Authorization endpoint

- ▶ Service provided by the **OAuth server**
  - ♦ Authenticates the resource owner (the user)
  - ♦ Asks for the delegation of access rights to its protected resources to the client
  - ♦ Send an authorization grant to the **redirection endpoint**

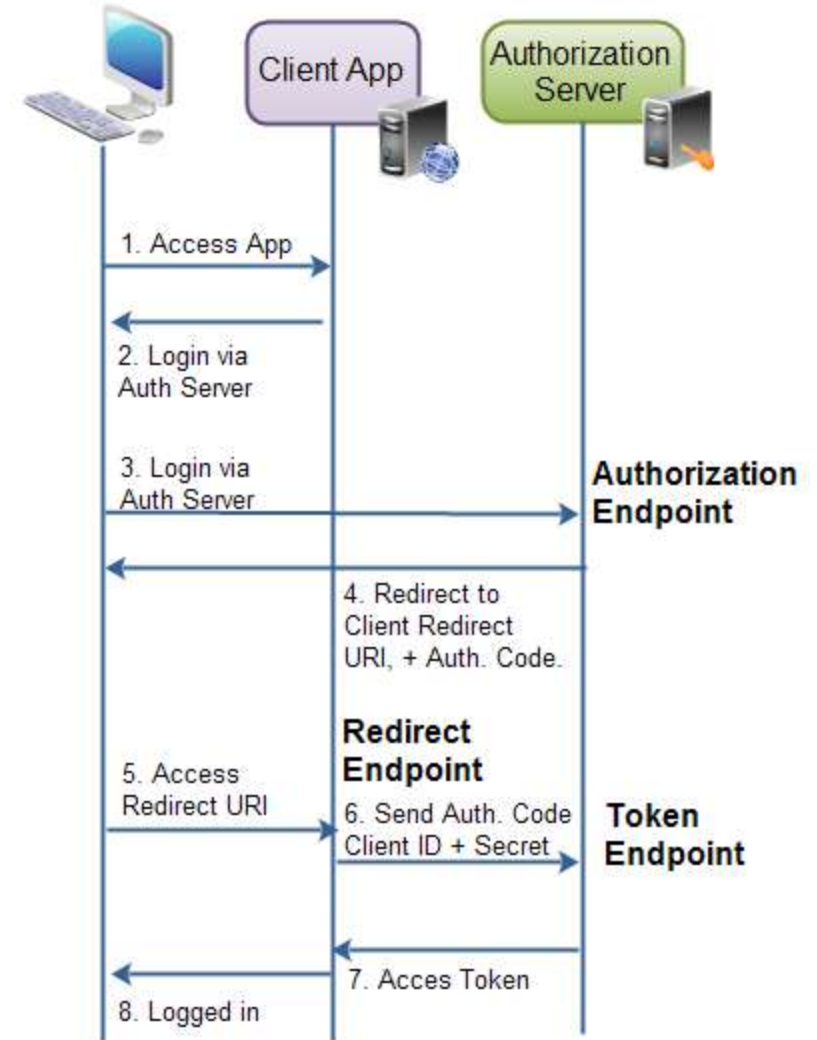
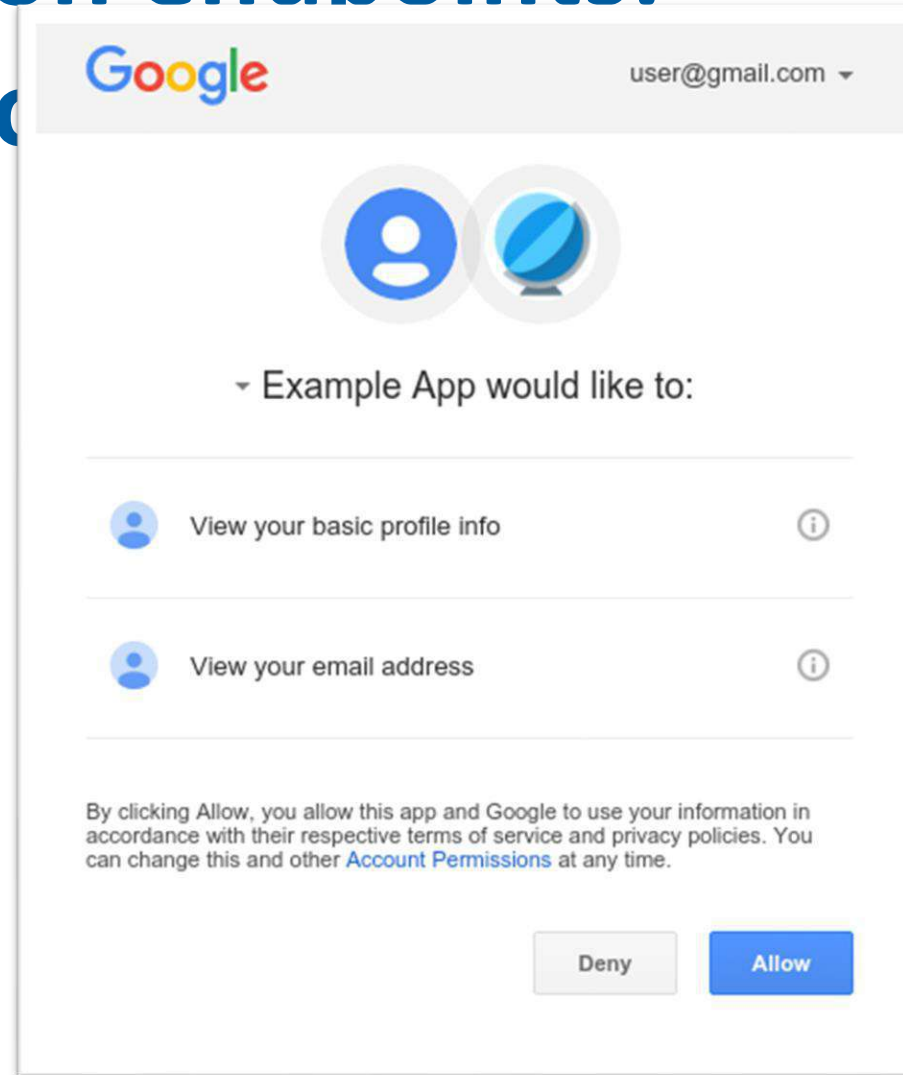


Image: <https://jenkov.com/tutorials/oauth2/endpoints.html>

# Communication endpoints: Authorization



# Communication endpoints: Token endpoint

- ▶ Service provided by the **OAuth server**
  - ◆ Produces access tokens given an authorization grant
  - ◆ It can also produce refresh tokens
  - ◆ Refresh tokens can be used to get new tokens
    - With an authorization grant
- ▶ Client authentication
  - ◆ ClientID + ClientSecret + HTTP basic authentication

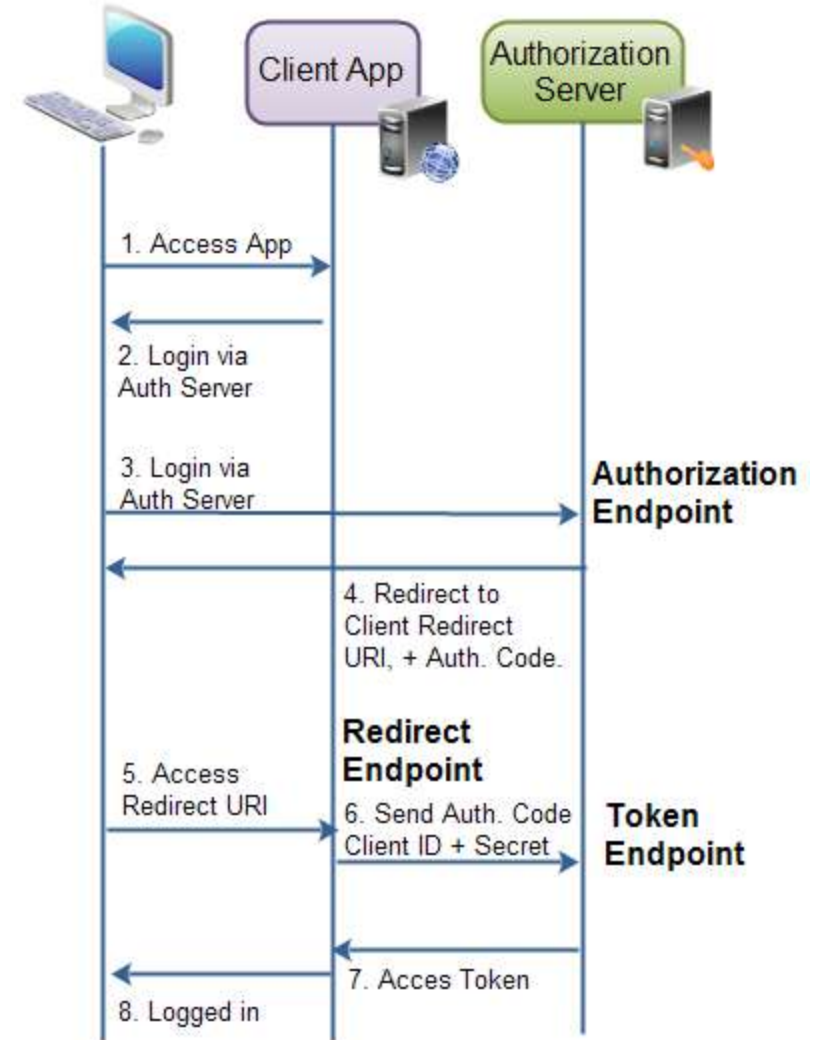


Image: <https://jenkov.com/tutorials/oauth2/endpoints.html>

# Communication endpoints: Redirect endpoint

- ▶ Service provided by the client
  - ♦ It collects the authorization grant provided by the OAuth server
  - ♦ It should be called by the OAuth server using an HTTP redirect

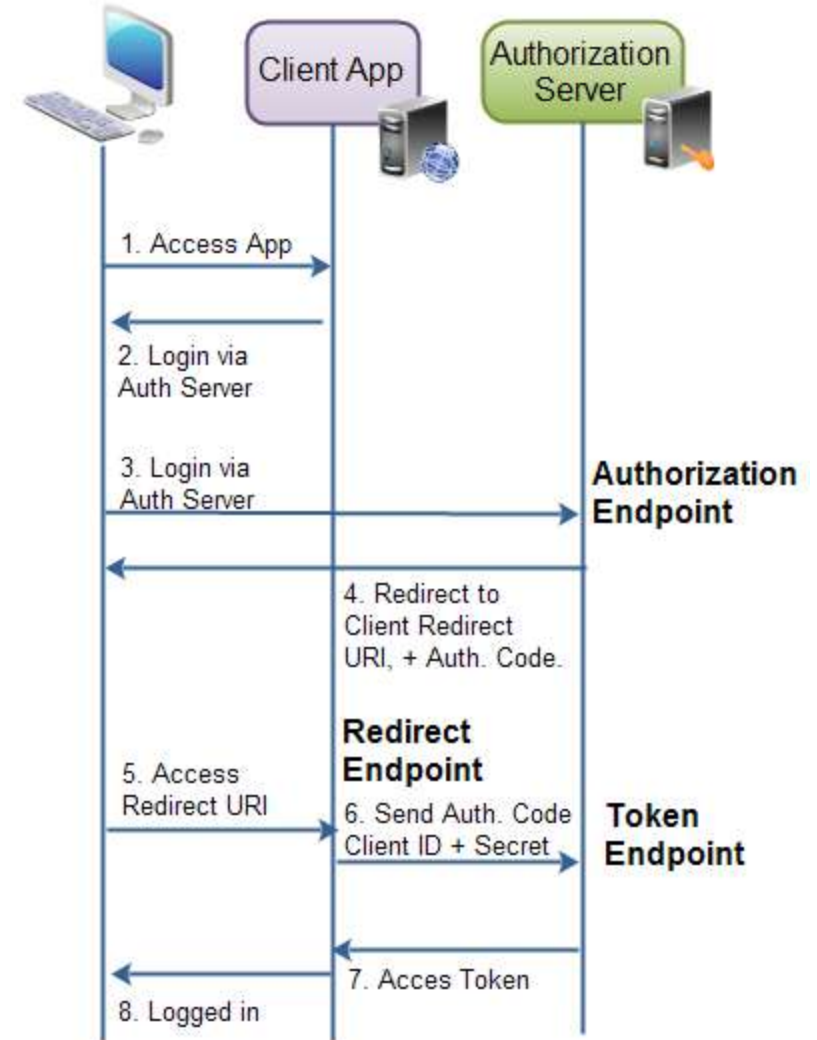


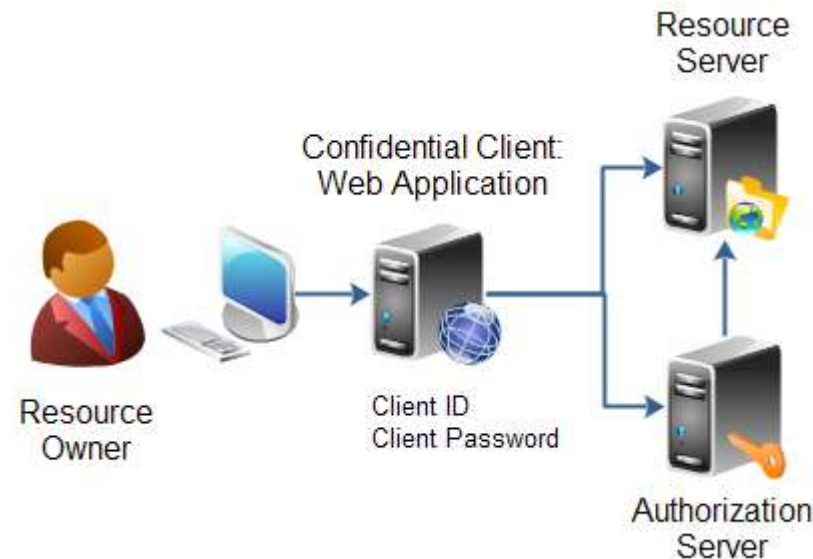
Image: <https://jenkov.com/tutorials/oauth2/endpoints.html>

# Application (client) types

- ▷ Type is related with the ability to maintain the confidentiality of client credentials
  - ♦ Even from the resource owner
- ▷ Confidential
  - ♦ Capable
  - ♦ e.g. a secure server
- ▷ Public
  - ♦ Incapable
  - ♦ e.g. a web browser-based application, a mobile App
- ▷ Different application types will be allowed to execute different flows

# Application (client) profiles

- ▶ Web application
  - ◆ Confidential client running on a web server

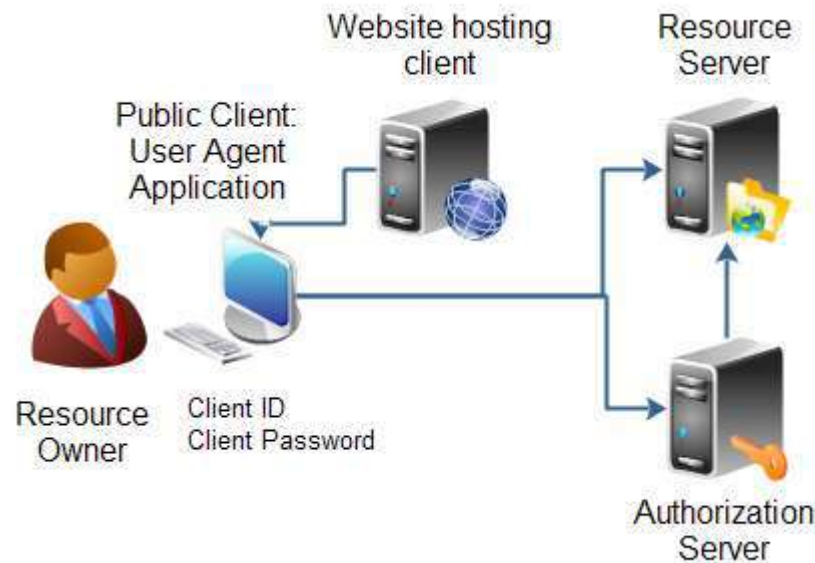


<https://jenkov.com/tutorials/oauth2/client-types.html>

# Application (client) profiles

## ▷ User-agent based application

- ♦ Public client where the client code runs on a user-agent application
  - e.g. a browser



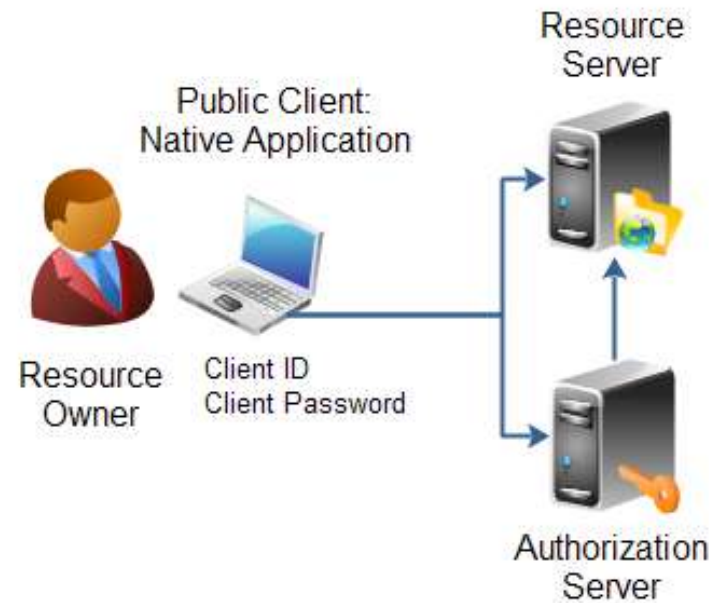
<https://jenkov.com/tutorials/oauth2/client-types.html>



# Application (client) profiles

## ▷ Native application

- ♦ Public client installed and executed on the device used by the resource owner



<https://jenkov.com/tutorials/oauth2/client-types.html>

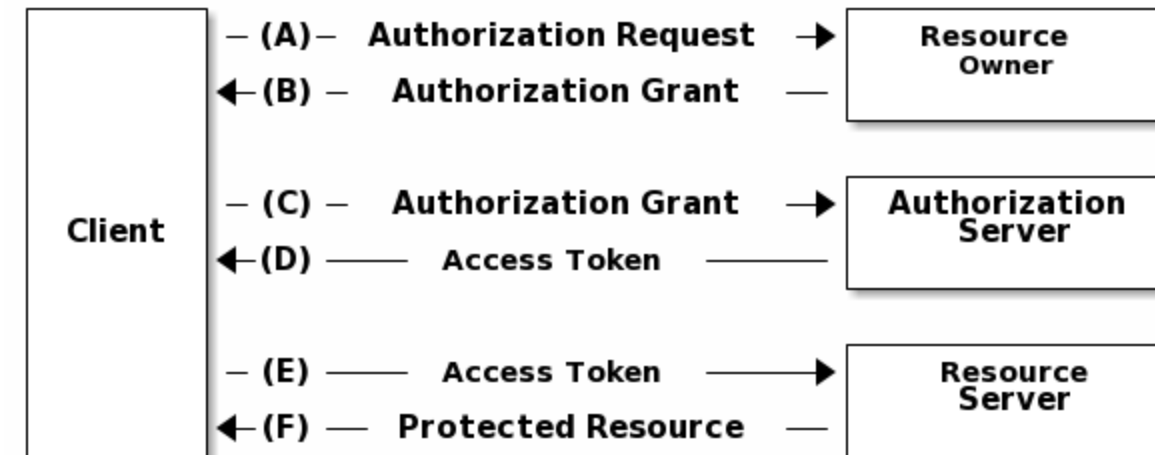
# Application (client) registration (in an OAuth server)

- ▷ Clients accessing OAuth servers must be previously registered
  - ◆ Nevertheless, the standard does not exclude unregistered clients
  - ◆ A registered client is given a unique identifier
    - ClientID
- ▷ Registration includes both informational, legal and operational information
  - ◆ Redirection URLs
  - ◆ Acceptance of legal terms
  - ◆ Application (client) name, logo, web site, description
  - ◆ Client type
  - ◆ Client authentication method (for confidential clients)

# OAuth tokens:

## Authorization grant

- ▷ Created by an OAuth server
  - ♦ Upon authenticating a resource owner and getting its consent to grant access to a protected resource
  - ♦ An opaque byte blob that makes sense only to its issuer
- ▷ Short validity time
  - ♦ Just enough to get an access token



# OAuth tokens:

## Access token

- ▶ Created by an OAuth server
  - ♦ Upon authenticating a client and receiving an authorization grant
  - ♦ An opaque byte blob that makes sense to its issuer and to the resource owner
    - An access capability
- ▶ Bearer tokens
  - ♦ Clients need to protect their use with HTTPS
  - ♦ Clients can handover tokens to others



# OAuth tokens:

## Refresh token

- ▷ Created by an OAuth server
  - ♦ When creating an access token
  - ♦ An opaque byte blob that makes sense only to its issuer
  - ♦ It can be used to collect a new access token
    - Still requiring the client authentication
- ▷ Bearer tokens
  - ♦ Clients need to protect their use with HTTPS
  - ♦ Clients can handover tokens to others



# OAuth flows

- ▶ Authorization code flow
  - ♦ 3-legged OAuth
  - ♦ Default OAuth flow
  - ♦ The most secure
- ▶ Implicit flow (grant)
- ▶ Resource owner password credentials flow
- ▶ Client credentials flow
  - ♦ 2-legged flow

OAuth 2 flows	Needs front end	Needs back end	Has user interaction	Needs client secret
Authorization Code	✓	✓	✓	✓
Implicit Grant	✓	✗	✓	✗
Client Credentials	✗	✓	✗	✓
Password Grant	✓	✓	✓	✓

# Authorization code flow

- ▷ 3-legged OAuth
  - ◆ Enables checking the identity of the 3 involved actors
- ▷ OAuth server authenticates the resource owner
  - ◆ Username + password or other means
- ▷ OAuth server authenticates the client
  - ◆ ClientID + ClientSecret + HTTP basic authorization
- ▷ Client authenticates the OAuth server
  - ◆ Certificate + URL

OAuth 2 flows	Needs front end	Needs back end	Has user interaction	Needs client secret
Authorization Code	✓	✓	✓	✓

# Authorization code flow

## ► Requirements

- ◆ Confidential application types
- ◆ Secure storage for tokens, ClientID and ClientSecret

## ► Setup

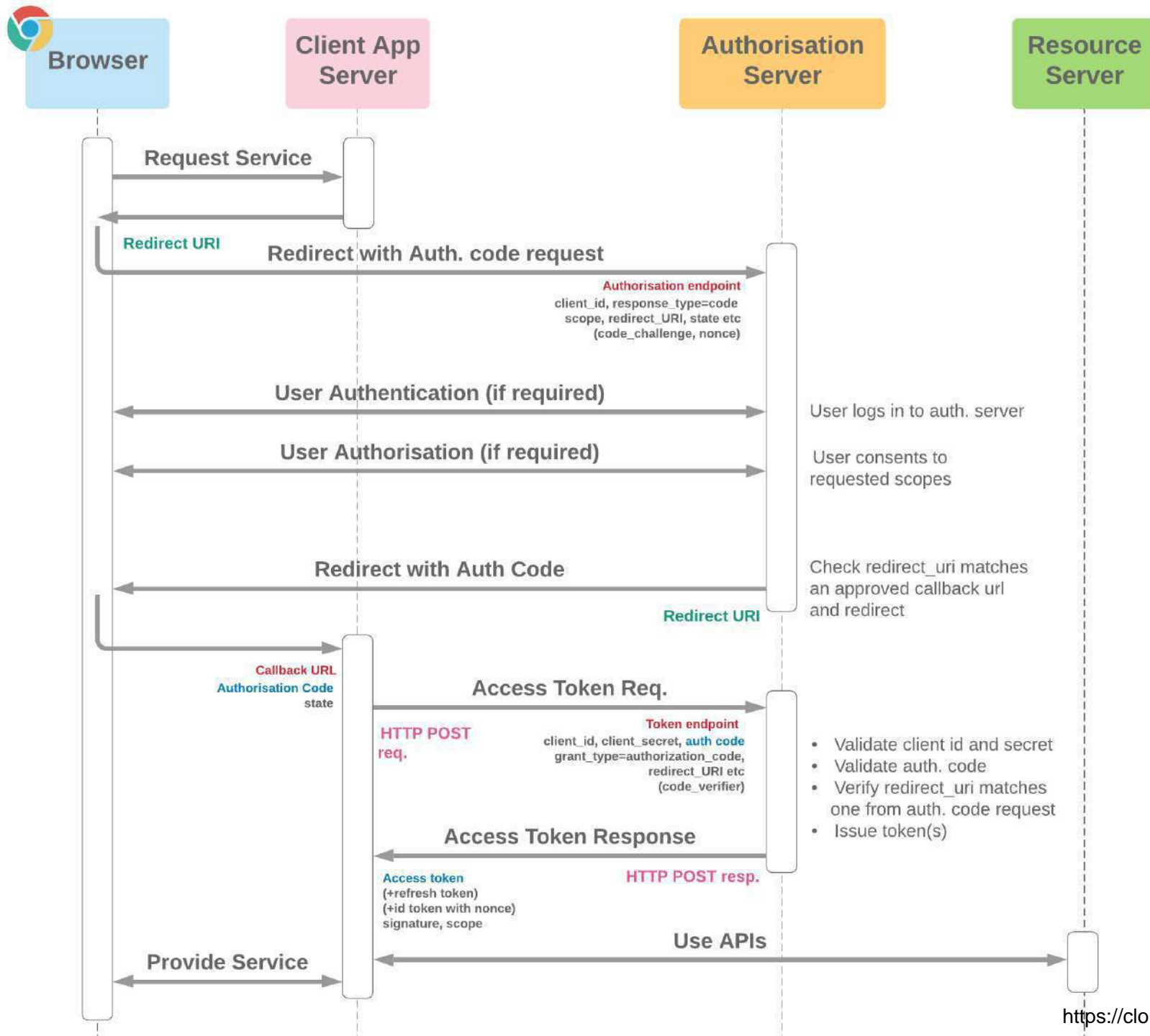
- ◆ Client registration in the OAuth server
  - Client receives ClientID and ClientSecret
  - Not regulated by OAuth

OAuth 2 flows	Needs front end	Needs back end	Has user interaction	Needs client secret
Authorization Code	✓	✓	✓	✓



# Authorization code flow

- ▷ Resource owner uses a server-based Web App
  - ♦ The client
- ▷ The client uses the resource server API to get a resource
  - ♦ The resource server redirects the client to the OAuth server
- ▷ The OAuth server authenticates the resource owner
  - ♦ And sends an authorization grant to the client
- ▷ The client gets an access token from the OAuth server
  - ♦ Using its credentials (to have access permission)
  - ♦ Using its authorization grant
- ▷ The client uses again the resource server API to get a resource
  - ♦ This time providing an access token



# Implicit flow

## ▷ Requirements

- ♦ Public application types

## ▷ Setup

- ♦ Client registration in the OAuth server
  - Client receives ClientID
  - Not regulated by OAuth

## ▷ Limitations

- ♦ No client authentication
- ♦ No refresh tokens

OAuth 2 flows	Needs front end	Needs back end	Has user interaction	Needs client secret
Implicit Grant	✓	✗	✓	✗

# Implicit flow

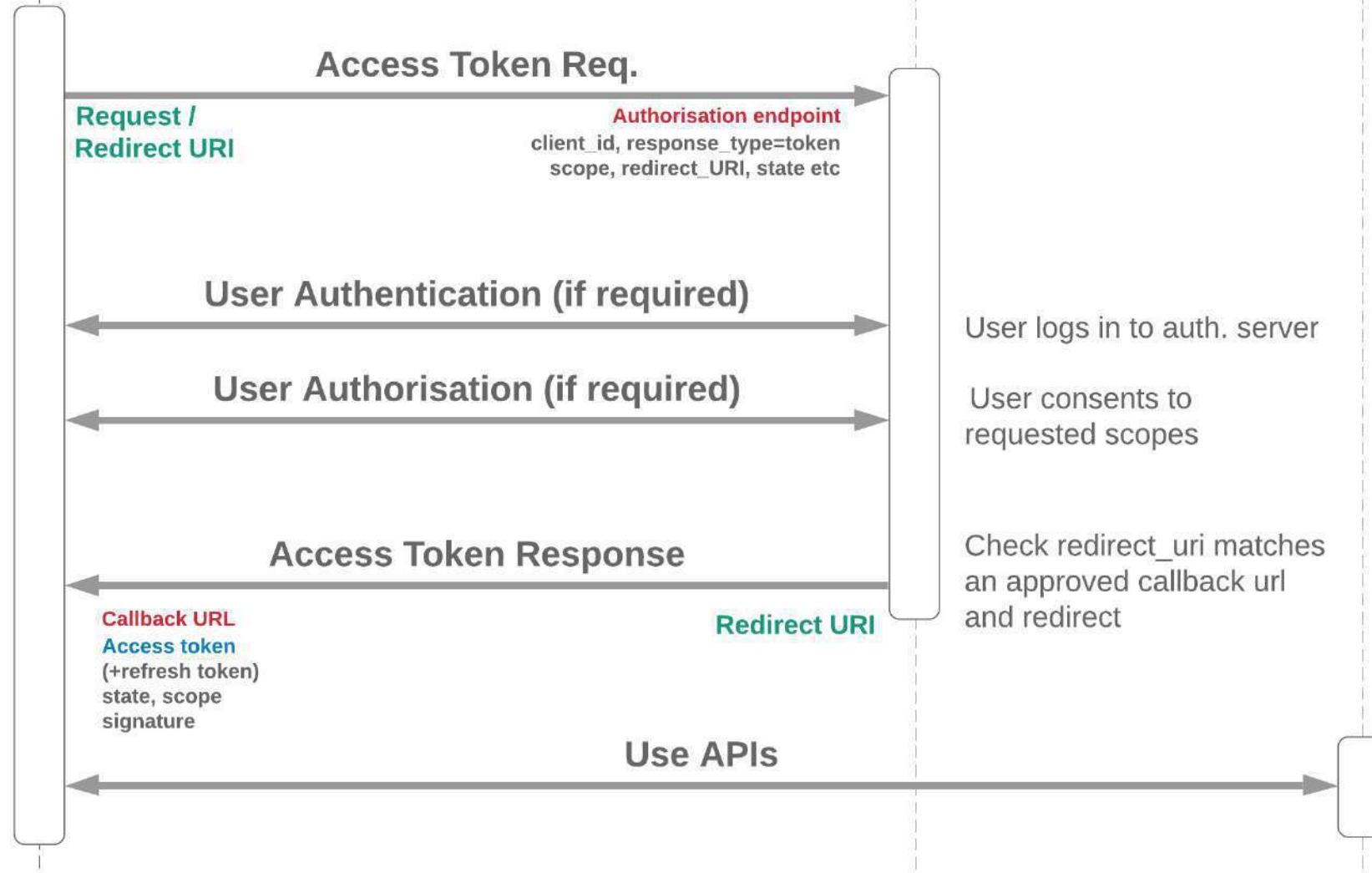
- ▷ Resource owner uses a mobile or client-based Web App
  - ♦ The client
- ▷ The client uses the resource server API to get a resource
  - ♦ The resource server redirects the client to the OAuth server
- ▷ The OAuth server authenticates the resource owner
  - ♦ And sends an access token to the client
- ▷ The client uses again the resource server API to get a resource
  - ♦ This time providing an access token



Browser

Authorisation  
Server

Resource  
Server



# Resource owner password flow

## ▷ Requirements

- ◆ Confidential application types
- ◆ Sharing of resource owner credentials with client applications
- ◆ Secure storage for tokens, ClientID and ClientSecret

## ▷ Setup

- ◆ Client registration in the OAuth server
  - Client receives ClientID and ClientSecret
  - Not regulated by OAuth

## ▷ Limitations

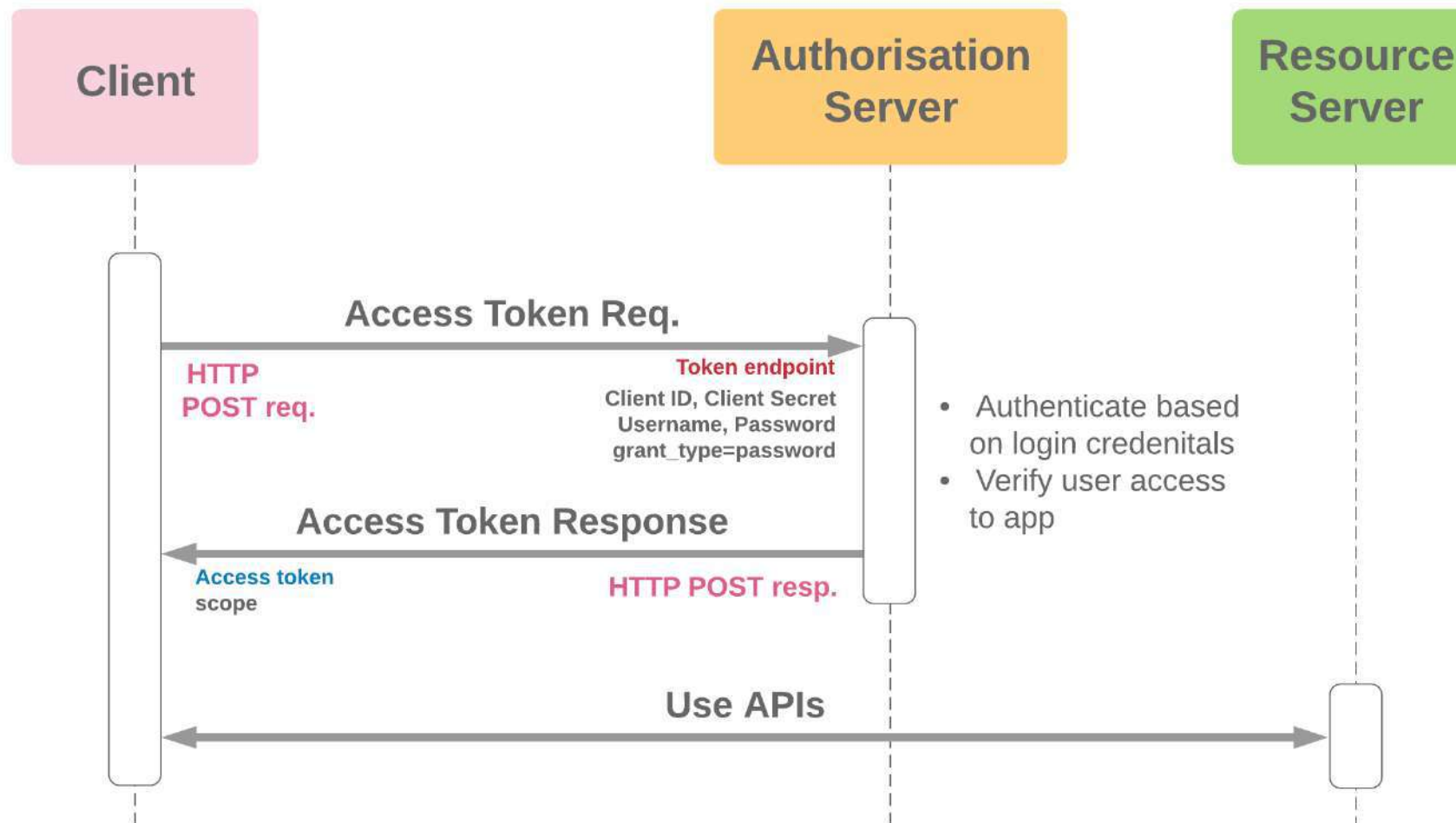
- ◆ Resource owners need to trust on client applications

OAuth 2 flows	Needs front end	Needs back end	Has user interaction	Needs client secret
Password Grant	✓	✓	✓	✓

# Resource owner password flow

- ▷ Resource owner uses a server-based Web App
  - ♦ The client
- ▷ The client uses the resource server API to get a resource
  - ♦ The resource server requests a token
- ▷ The client asks the resource owner for authentication credentials
- ▷ The client gets an access token from the OAuth server
  - ♦ Using its credentials (to have access permission)
  - ♦ Using the resource owner's credentials
  - ♦ These should be immediately discarded
- ▷ The client uses again the resource server API to get a resource
  - ♦ This time providing an access token

OAuth 2 flows	Needs front end	Needs back end	Has user interaction	Needs client secret
Password Grant	✓	✓	✓	✓





# Client credentials flow

## ▷ Requirements

- ◆ Confidential application types
- ◆ Secure storage for tokens, ClientID and ClientSecret

## ▷ Setup

- ◆ Client registration in the OAuth server
  - Client receives ClientID and ClientSecret
  - Not regulated by OAuth

## ▷ Limitations

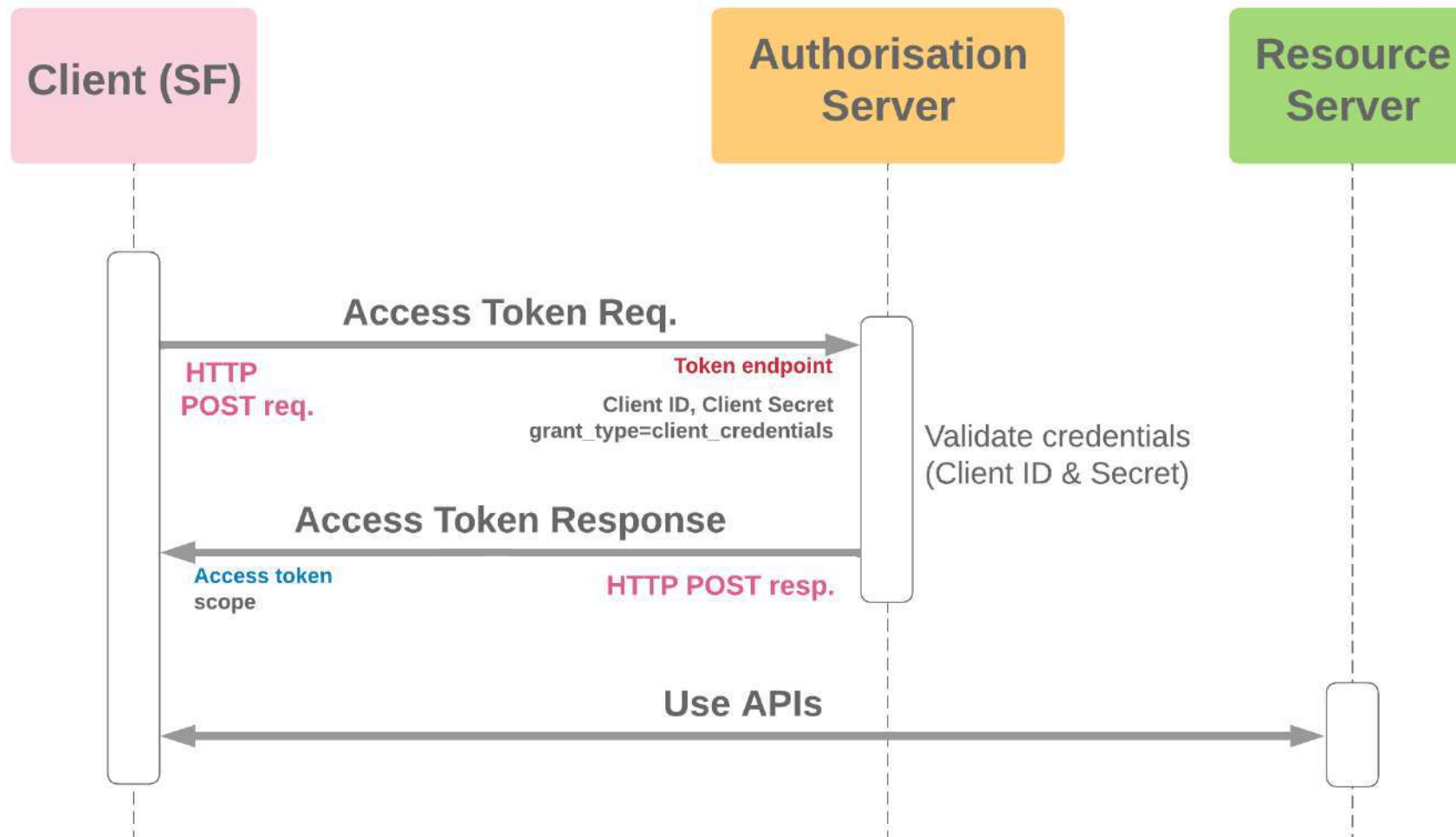
- ◆ No resource owner authentications or authorizations

OAuth 2 flows	Needs front end	Needs back end	Has user interaction	Needs client secret
Client Credentials	✗	✓	✗	✓

# Client credentials flow

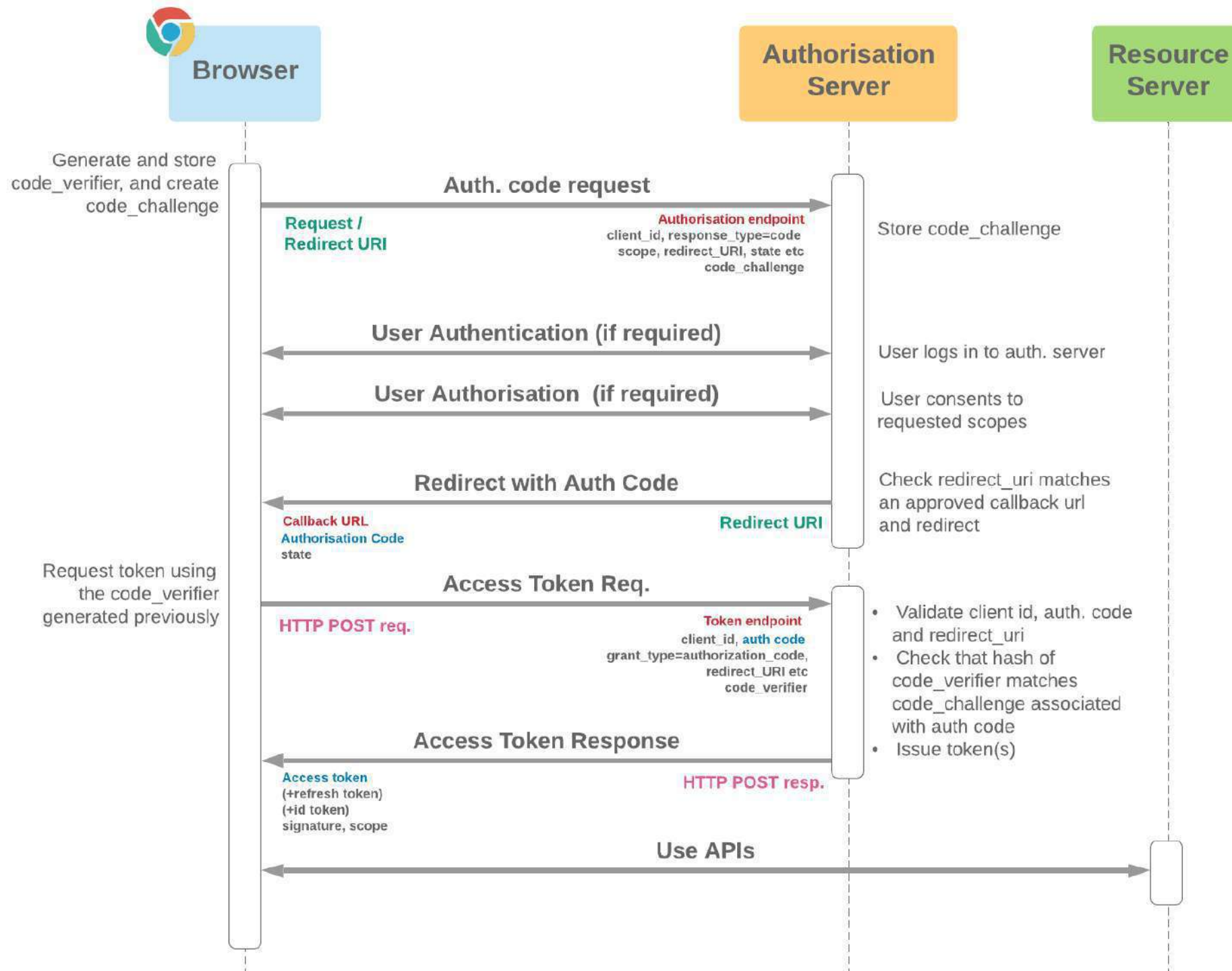
- ▶ Resource owner uses a server-based Web App
  - ♦ The client
- ▶ The client uses the resource server API to get a resource
  - ♦ The resource server requests a token
- ▶ The client gets an access token from the OAuth server
  - ♦ Using its credentials (to have access permission)
- ▶ The client uses again the resource server API to get a resource
  - ♦ This time providing an access token

OAuth 2 flows	Needs front end	Needs back end	Has user interaction	Needs client secret
Client Credentials	✗	✓	✗	✓



# Proof Key for Code Exchange (PKCE, RFC 7636)

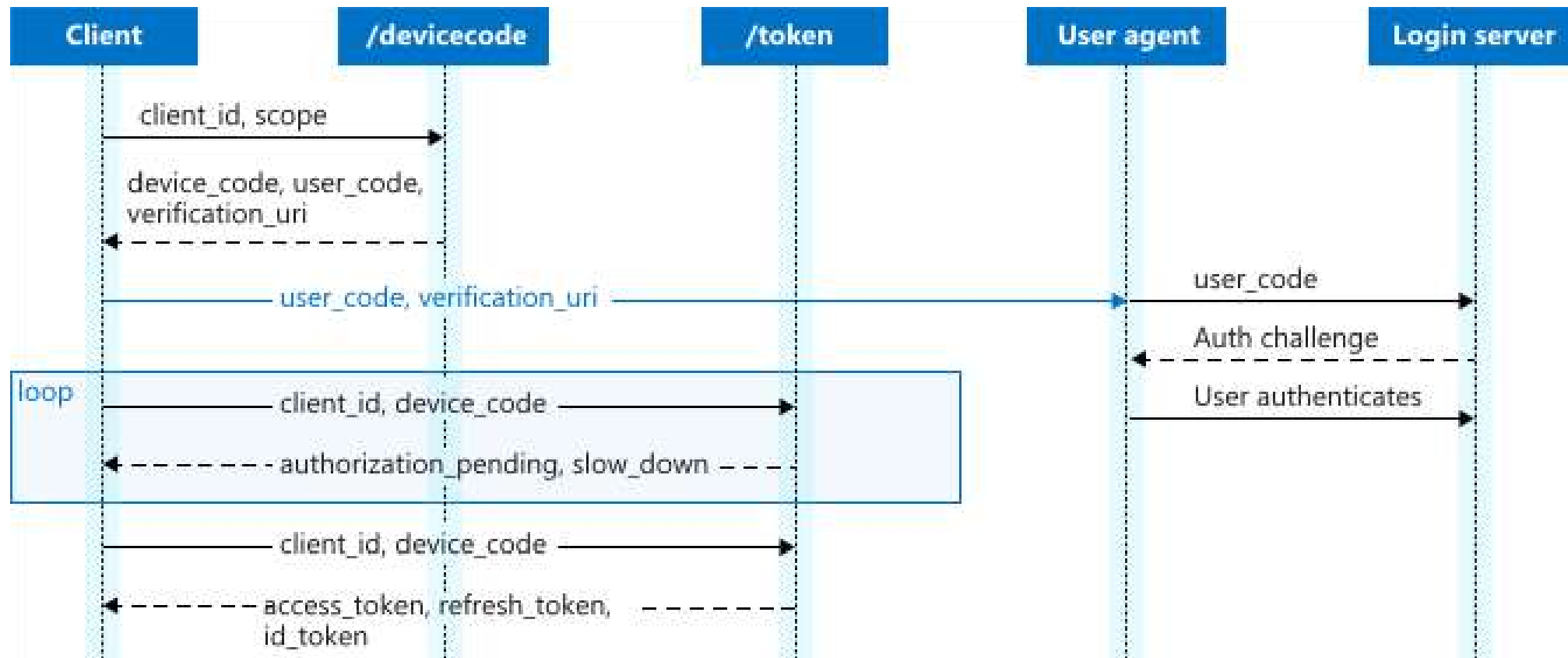
- ▷ Binds authorization grants to their requesters
  - ◆ Using a Code Challenge
    - A digest of a Code Verifier
    - A bit commitment
  - ◆ Cannot be used by eavesdroppers
- ▷ The requester is required to demonstrate the ownership of the authorization grant when fetching the access token
  - ◆ Providing the Code Verifier



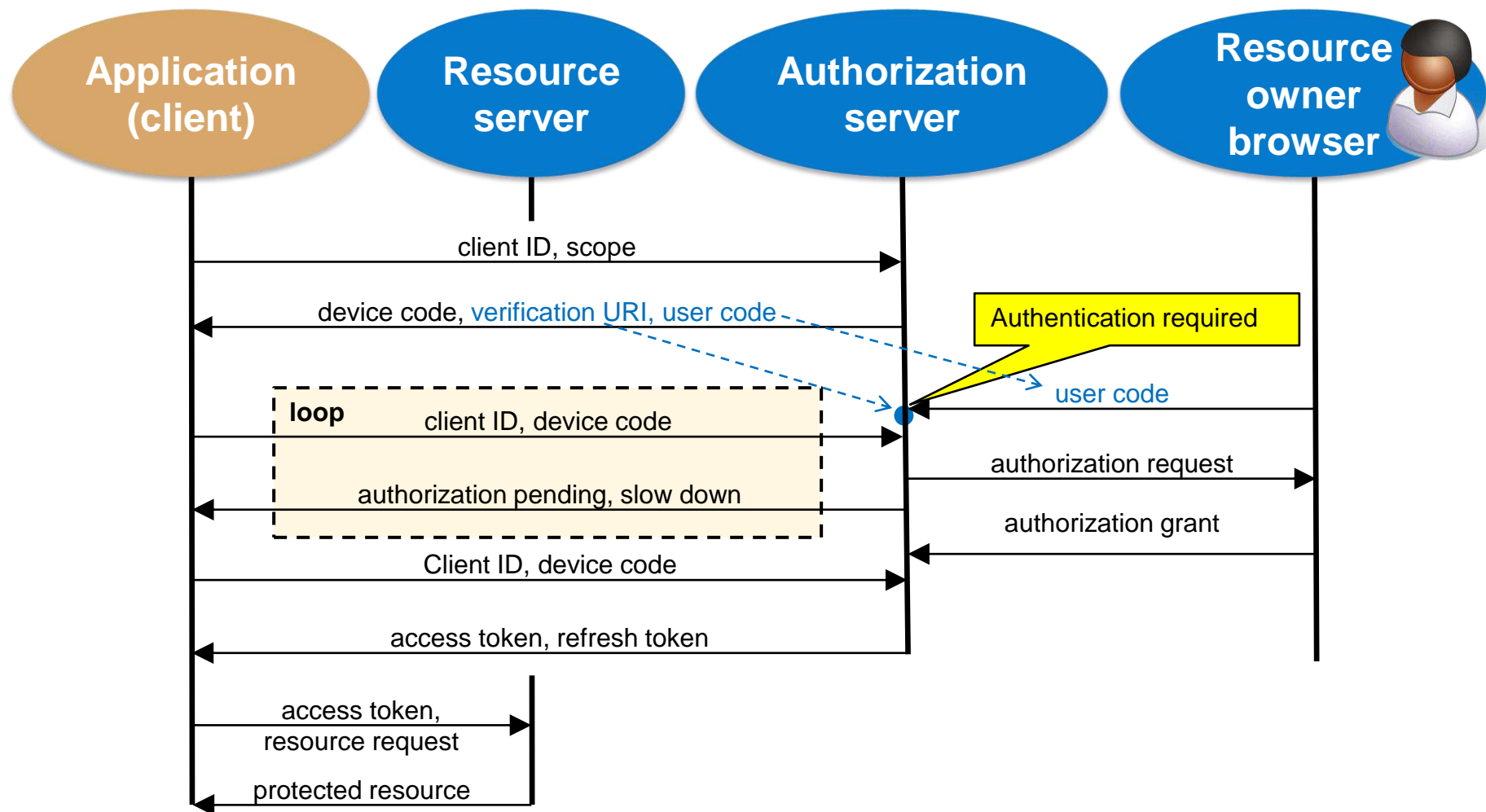
# Device authorization grant (RFC 8628)

- ▷ In some cases the user is using a device with no browser to interact with a OAuth client
  - ◆ No HTTP redirections to Authorization server and back to client
  - ◆ No user interface
    - To authenticate the user
    - To review and authorize request
- ▷ Solution
  - ◆ Use a second device to perform the user authentication and to grant the authorization
    - e.g. mobile phone, tablet, etc.
  - ◆ Client fetches the access token from the Authorization server
    - Possibly with a refresh token

# Device authorization grant (RFC 8628)



# Actual protocol flow





# Linux security mechanisms

# Mechanisms

- ▷ Capabilities
- ▷ cgroups (control groups)
- ▷ LSM (Linux Security Modules)

# Linux management privileges

## ▷ Initial UNIX philosophy

- ♦ Privileged processes (UID = 0)
  - Bypass all kernel permission checks
- ♦ Unprivileged processes (UID  $\neq$  0)
  - Subject to permission checking based on their credentials
  - Effective UID, effective GID, secondary group list

# Unix file protection ACLs:

## Special protection bits

### ▷ Set-UID bit

- ♦ Is used to change the UID of processes executing the file

```
creator:Pictures$ ls -la /usr/bin/passwd  
-rwsr-xr-x 1 root root 59640 Mar 22 2019 /usr/bin/passwd
```

### ▷ Set-GID bit

- ♦ Is used to change the UID of processes executing the file

```
creator:Pictures$ ls -la /usr/bin/at  
-rwsr-sr-x 1 daemon daemon 51464 Feb 20 2018 /usr/bin/at
```

### ▷ Sticky bit

- ♦ Hint to keep the file/directory as much as possible in memory cache

```
creator:Pictures$ ls -la /tmp  
total 108  
drwxrwxrwt 25 root root 4096 Dec 15 13:12 .
```

# Privilege elevation:

## Set-UID mechanism

- ▷ Change the effective UID of a process running a program stored on a Set-UID file
  - ♦ If a program file is owned by UID **X** and the set-UID bit of its ACL is set, then it will be executed in a process with UID **X**
    - Independently of the UID of the subject that executed the program
- ▷ Allows normal users to execute privileged tasks encapsulated in administration programs
  - ♦ Change the user's password (**passwd**)
  - ♦ Change to super-user mode (**su**, **sudo**)
  - ♦ Mount devices (**mount**)

# Privilege elevation:

## Set-UID mechanism (cont.)

### ▷ Effective UID / Real UID

- ♦ **Real UID (rUID)** is the UID of the process creator
  - App launcher
- ♦ **Effective UID (eUID)** is the UID of the process
  - The one that really matters for defining the rights of the process
  - eUID may differ from rUID

### ▷ UID change

- ♦ **Ordinary application**
  - eUID = rUID = UID of process that executed **exec**
  - eUID cannot be changed (unless = 0)
- ♦ **Set-UID application**
  - eUID = UID of **exec**'d application file, rUID = initial process UID
  - eUID can revert to rUID
- ♦ **rUID cannot change**

# Privilege elevation:

## Set-UID/Set-GID decision flowchart

### ▷ exec ( path, ...)

- ♦ File referred by path has Set-UID?
  - ♦ Yes
    - ID = path owner
    - Change the process effective UID to ID of path owner
  - ♦ No
    - Do nothing
- ♦ File referred by path has Set-GID?
  - ♦ Yes
    - ID = path GID
    - Change the process GIDs to ID only
  - ♦ No
    - Do nothing

# Capabilities

- ▷ Protection mechanism introduced in Kernel 2.2
- ▷ Allow to divide the traditional super-user privileges into distinct units
  - ♦ That can be independently enabled and disabled
- ▷ Capabilities are a per-thread attribute
  - ♦ Propagated through forks
  - ♦ Changed explicitly of by execs



# List of capabilities:

## Examples (small sample ...)

- ▷ CAP\_CHOWN
  - ♦ Make arbitrary changes to file UIDs and GIDs
- ▷ CAP\_DAC\_OVERRIDE / CAP\_DAC\_READ\_SEARCH
  - ♦ Bypass file permission / directory transversal checks
- ▷ CAP\_KILL
  - ♦ Bypass permission checks for sending signals
- ▷ CAP\_NET\_ADMIN
  - ♦ Perform various network-related operations
- ▷ CAP\_SYS\_ADMIN
  - ♦ Overloaded general-purpose administration capability

```
$ capsh --explain=CAP_NET_ADMIN
cap_net_admin (12) [/proc/self/status:CapXXX: 0x00000000000001000]
```

Allows a process to perform network configuration operations:

- interface configuration
- administration of IP firewall, masquerading and accounting
- setting debug options on sockets
- modification of routing tables
- setting arbitrary process, and process group ownership on sockets
- binding to any address for transparent proxying (this is also allowed via CAP\_NET\_RAW)
- setting TOS (Type of service)
- setting promiscuous mode
- clearing driver statistics
- multicasting
- read/write of device-specific registers
- activation of ATM control sockets

# Capability management

- ▷ Per-thread capabilities
  - ♦ They define the privileges of the thread
  - ♦ Divided in **sets**
- ▷ Sets
  - ♦ Effective
  - ♦ Inheritable
  - ♦ Permitted
  - ♦ Bounding
  - ♦ Ambient

# Thread capability sets:

## Effective

- ▶ Set of capabilities used by the kernel to perform permission checks for the thread
- ▶ That is: these are the effective capabilities being used

# Thread capability sets:

## Inheritable

- ▷ Set of capabilities preserved across an **exec**
  - ◆ Remain inheritable for any program
- ▷ Are added to the permitted set when executing a program that has the corresponding bits set in the file inheritable set

# Thread capability sets:

## Permitted

### ▷ Limiting superset

- ◆ For the effective capabilities that the thread may assume
- ◆ For the capabilities that may be added to the inheritable set
  - Except for threads w/ CAP\_SETPCAP in their effective set

### ▷ Once dropped, it can never be reacquired

- ◆ Except upon executing a file with special capabilities

```
$ getcap /bin/*  
/bin/ping cap_net_raw=ep
```

# Thread capability sets:

## Bounding

- ▶ Set used to limit the capabilities that are gained during an exec
  - ◆ From a file with capabilities set
- ▶ Was previously a system-wide attribute
  - ◆ Now is a per-thread attribute

# Thread capability sets:

## Ambient

- ▶ Set of capabilities that are preserved across an **exec** of an unprivileged program
  - ◆ No set-UID or set-GID
  - ◆ No capabilities set
- ▶ Executing a privileged program will clear the ambient set

# Thread capability sets:

## Ambient

- ▶ Ambient capabilities must be both permitted and inheritable
  - ♦ One cannot preserve something one cannot have
  - ♦ One cannot preserve something one cannot inherit
  - ♦ Automatically lowered if either of the corresponding permitted or inheritable capabilities is lowered
- ▶ Ambient capabilities are added to the permitted set and assigned to the effective set upon an **exec**



# Files extended attributes (xattr)

## ▷ Files' metadata in UNIX-like systems

- ♦ Some not interpreted by kernels

## ▷ Linux: key-value pairs

- ♦ Keys can be defined or undefined
- ♦ If defined, their value can be empty or not
- ♦ Key's namespaces
  - `namespace.attr_name[.attr_name]`

## ▷ Namespaces

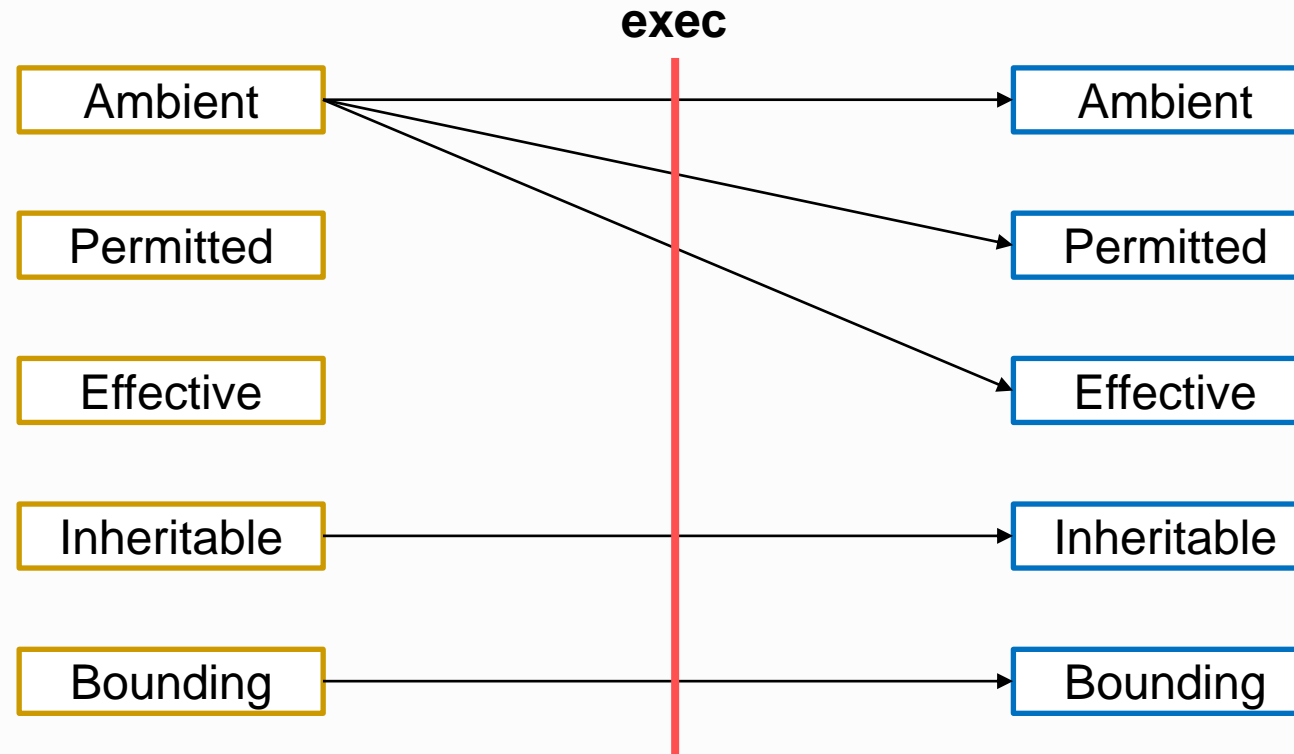
- ♦ **security**
  - For files' capabilities
  - `setcap` / `getcap`
- ♦ **system**
  - ACL
- ♦ **trusted**
  - Protected metadata
- ♦ **user**
  - `setfattr` / `lsattr` / `getfattr`

# File capabilities

- ▷ Stored in the `security.capability` attribute
- ▷ Specify capabilities for threads that exec a file
  - ◆ **Permitted set**
    - Immediately forced into the permitted set
    - Previous AND with the thread's bounding set
  - ◆ **Inheritable set**
    - To AND with the threads' inheritable set
    - Can be used to reduce the effective set upon the exec
  - ◆ **Effective bit**
    - Enforce all new capabilities into the thread's effective set

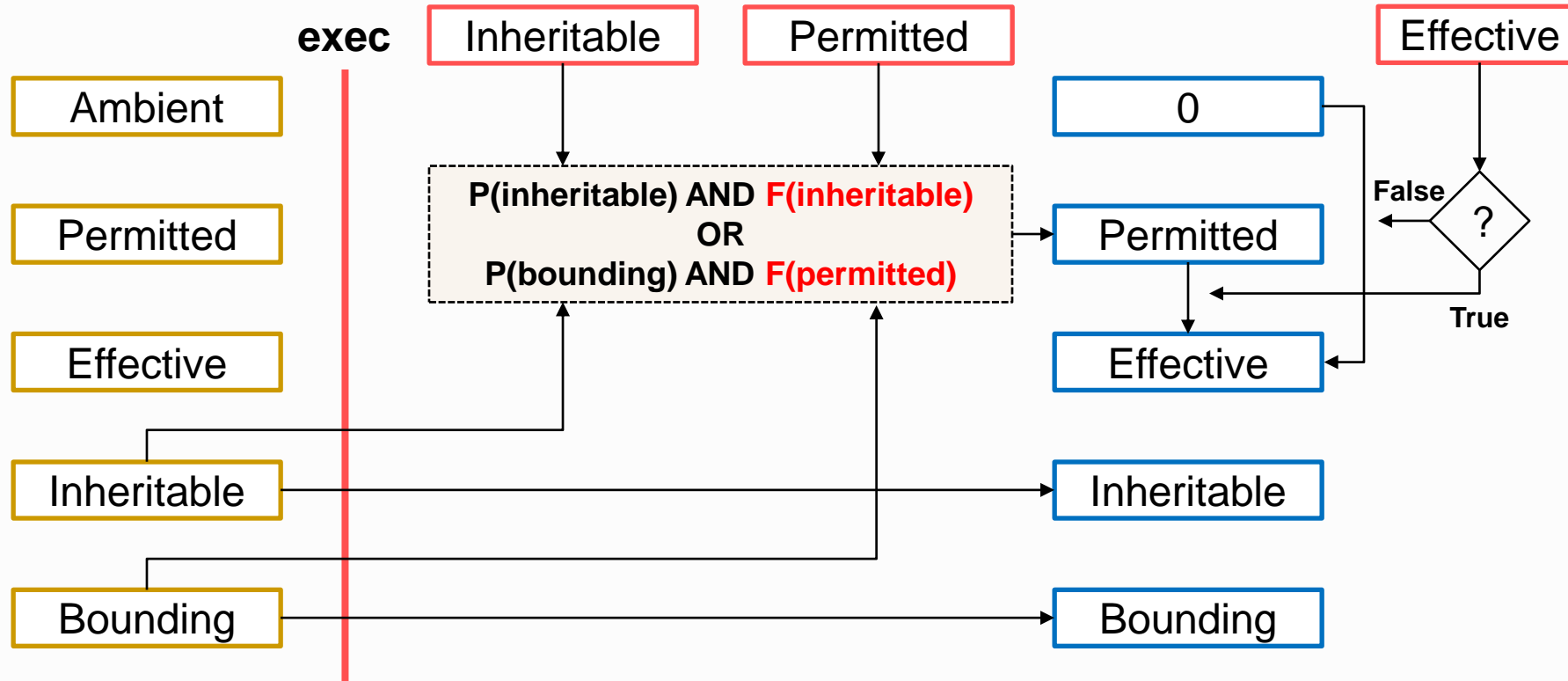
# Capability transfer across exec:

## No privileged files



# Capability transfer across exec (non-root)

## Privileged files



# Capability transfer across exec (root)

- ▷ EUID = 0 or RUID = 0
  - ♦ Capability sets are considered to be all 1's
- ▷ EUID = 0
  - ♦ File effective bit considered 1
- ▷ Exception: EUID = 0, RUID  $\neq$  0
  - ♦ Set-UID file was executed
  - ♦ File capabilities are honored if present

# Control groups (cgroups)

- ▷ Collection of processes bound by the same criteria and associated with a set of parameters or limits
- ▷ cgroups are organized hierarchically
  - ♦ cgroup file system
  - ♦ Limits can be defined at each hierarchical level
    - Affecting the sub-hierarchy underneath
- ▷ Subsystems
  - ♦ Kernel component that modifies the behavior of cgroup processes
  - ♦ Resource controllers (or simply controllers)

# cgroups v1 and v2

- ▷ Currently two versions coexist
  - ♦ But controllers can only be used in one of them

# cgroups file system

- ▷ This file system is created by mounting several controllers as cgroup-type file system entities
  - ♦ Usually `/sys/fs/cgroup`
  - ♦ In V2 all controllers are part of a single cgroup2
- ▷ Each controller defines a tree of cgroups below the mount point
  - ♦ e.g. memory controller → `/sys/fs/cgroup/.../memory[...]`



# cgroup V2 (and V1) controllers

- ▷ cpu (cpu & cpuacct in V1)
  - ♦ CPU usage & accounting
- ▷ cpuset
  - ♦ CPU bounding
- ▷ memory
  - ♦ Memory usage & accounting
- ▷ devices
  - ♦ Device creation & usage
- ▷ freezer
  - ♦ Suspend/resume groups of processes
- ▷ io (blkio in V1)
  - ♦ Block I/O management
- ▷ perf\_event
  - ♦ Performance monitoring
- ▷ hugelb
  - ♦ Huge pages management
- ▷ pids
  - ♦ # of processes in cgroup
- ▷ rdma
  - ♦ RDMA / IB resources' management
- ▷ Deprecated from V1
  - ♦ net\_cls
    - Outbound packet classification
  - ♦ net\_prio
    - Network interfaces priorities

# cgroup V2 definition

- ▷ Directory under `/sys/fs/cgroup`
  - ◆ With a set of controllers defined by `cgroup.controllers`
  - ◆ With hierarchy limits defined by `cgroup.depth` and `cgroup.descendants`
  - ◆ With files to send KILL signals (`cgroup.kill`) and freeze/unfreeze orders (`cgroup.freeze`) to all cgroup processes
    - Including descendants
  - ◆ The processes using the cgroup are given by `cgroup.procs` and their status reported by `cgroups.events`
    - We can add a process to a cgroup just by writing its PID on the first file
- ▷ For each active controller, specific files will exist
- ▷ Processes can only belong to leaf cgroups
  - ◆ “No internal processes” rule

# cgroups of a process

- ▷ A process can be controlled by an arbitrary number of cgroups
- ▷ The list of a process' cgroups is given by the /proc file system
  - ♦ /proc/[PID]/cgroup

# Linux Security Modules (LSM)

- ▷ Framework to add new Mandatory Access Control (MAC) extensions to the kernel
- ▷ Those extensions are not kernel modules
  - ◆ They are embedded in the kernel code
  - ◆ They can be activated or not at boot time
  - ◆ List of extensions given by </sys/kernel/security/lsm>

# LSM extensions

- ▷ Capabilities (default)
- ▷ AppArmor
  - ◆ MAC for applications
- ▷ LoadPin
  - ◆ Kernel-loaded files origin enforcement
- ▷ SELinux
- ▷ Smack
  - ◆ Simplified Mandatory Access Control Kernel
- ▷ TOMOYO
  - ◆ Name-based MAC extension
- ▷ Yama
  - ◆ System-wide DAC security protections that are not handled by the core kernel itself
- ▷ SafeSetID
  - ◆ Restricts UID/GID transitions

Source: <https://www.kernel.org/doc/html/latest/admin-guide/LSM/index.html>

# AppArmor

- ▶ Enables the definition of per-application MAC policies
  - ◆ Profiles
  - ◆ Applications are identified by their path
    - Instead of i-node
- ▶ Profiles restrict applications' actions to the required set
  - ◆ All other actions will be denied
- ▶ Profiles define
  - ◆ Actions white-listed
  - ◆ Logging actions

# AppArmor: profiles

- ▷ Profiles are loaded into the kernel
  - ◆ Upon compilation from textual files
  - ◆ `apparmor_parser`
- ▷ Profiles can be used on a voluntary basis
  - ◆ `aa-exec`

# Confinement: Namespaces

- ▷ Allows partitioning of resources in views (namespaces)
  - ♦ Processes in a namespace have a restricted view of the system
  - ♦ Activated through syscalls by a simple process:
    - clone: Defines a namespace to migrate the process to
    - unshare: disassociates the process from its current context
    - setns: puts the process in a Namespace
- ▷ Types of Namespaces
  - ♦ **Mount**: Applied to mount points
  - ♦ **process id**: first process has id 1
  - ♦ **network**: "independent" network stack (routes, interfaces...)
  - ♦ **IPC**: methods of communication between processes
  - ♦ **uts**: name independence (DNS)
  - ♦ **user id**: segregation of permissions
  - ♦ **cgroup**: limitation of resources used (memory, cpu...)



## ## Create netns named mynetns

```
root@vm: ~# ip netns add mynetns
```

## ## Change iptables INPUT policy for the netns

```
root@linux: ~# ip netns exec mynetns iptables -P INPUT DROP
```

## ## List iptables rules outside the namespace

```
root@linux: ~# iptables -L INPUT
```

```
Chain INPUT (policy ACCEPT)
```

target	prot	opt	source	destination
--------	------	-----	--------	-------------

## ## List iptables rules inside the namespace

```
root@linux: ~# ip netns exec mynetns iptables -L INPUT
```

```
Chain INPUT (policy DROP)
```

target	prot	opt	source	destination
--------	------	-----	--------	-------------

## ## List Interfaces in the namespace

```
root@linux: ~# ip netns exec mynetns ip link list
1: lo: <LOOPBACK> mtu 65536 qdisc noop state DOWN mode DEFAULT group default qlen 100
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
```

## ## Move the interface enp0s3 to the namespace

```
root@linux: ~# ip link set enp0s3 netns mynetns
```

## ## List interfaces in the namespace

```
root@linux: ~# ip netns exec mynetns ip link list
1: lo: <LOOPBACK> mtu 65536 qdisc noop state DOWN mode DEFAULT group default qlen 100
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: enp0s3: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN mode DEFAULT...
    link/ether 08:00:27:83:0a:55 brd ff:ff:ff:ff:ff:ff
```

## ## List interfaces outside the namespace

```
root@linux: ~# ip link list
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN mode DEFAULT...
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
```

# Confinement: Containers

- ▷ Explores namespaces to provide a virtual view of the system
  - ◆ Network isolation, cgroups, user ids, mounts, etc...
- ▷ Processes are executed under a container
  - ◆ Container is an applicational construction and not of the core
  - ◆ Consists of an environment by composition of namespaces
  - ◆ Requires building bridges with the real system network interfaces, proxy processes
- ▷ Relevant approaches
  - ◆ **Linux Containers**: focus on a complete virtualized environment
    - evolution of OpenVZ
  - ◆ **Docker**: focus on running isolated applications based on a portable packet between systems
    - uses LXC

# Authentication protocols

# Identity attributes

## ▷ Set of attributes for setting apart individuals

- ♦ Name
- ♦ Numerical identifiers
  - Fixed for life
  - Variable with context
- ♦ Address
- ♦ Photo
- ♦ Identity of relatives
  - Usually parents
- ♦ ...

# Authentication: Definition

- ▷ Proof that an entity has a claimed identity attribute
  - Hi, I'm Joe
  - Prove it!
  - Here are my Joe's credentials
  - Credentials accepted/not accepted
- Hi, I'm over 18
  - Prove it!
  - Here is the proof
  - Proof accepted/not accepted

# Authentication: proof types

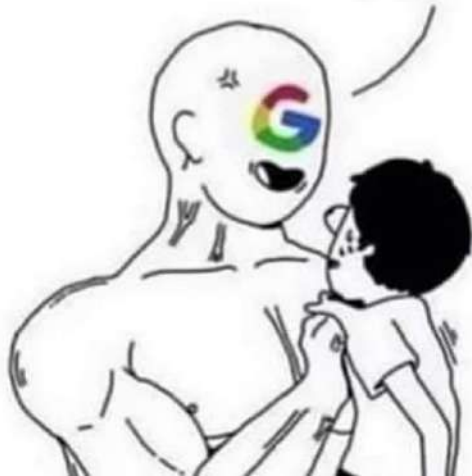
- ▷ Something we know
  - ♦ A secret memorized (or written down...) by Joe
- ▷ Something we have
  - ♦ An object/token solely held by Joe
- ▷ Something we are
  - ♦ Joe's Biometry
- ▷ Multi-factor authentication
  - ♦ Join or consecutive use of different proof types

# Multi-factor verification jokes

me: \*enters password  
correctly on new device\*

google:

Did you just sign in?





# Authentication: goals

- ▷ Authenticate interactors
  - ◆ People, services, servers, hosts, networks, etc.
- ▷ Enable the enforcement of authorization policies and mechanisms
  - ◆ Authorization  $\Rightarrow$  authentication
- ▷ Facilitate the exploitation of other security-related protocols
  - ◆ e.g. key distribution for secure communication

# Authentication: requirements

## ▷ Trustworthiness

- ◆ How good is it in proving the identity of an entity?
- ◆ How difficult is it to be deceived?
- ◆ Level of Assurance (LoA) (NIST, eIDAS, ISO 29115)
  - LoA 1 - Little or no confidence in the asserted identity
  - LoA 2 - Some confidence in the asserted identity
  - LoA 3 - High confidence in the asserted identity
  - LoA 4 - Very high confidence in the asserted identity

## ▷ Secrecy

- ◆ No disclosure of secrets used by legitimate entities

# Authentication: requirements

## ▷ Robustness

- ◆ Prevent attacks to the protocol data exchanges
- ◆ Prevent on-line DoS attack scenarios
- ◆ Prevent off-line dictionary attacks

## ▷ Simplicity

- ◆ It should be as simple as possible to prevent entities from choosing dangerous shortcuts

## ▷ Deal with vulnerabilities introduced by people

- ◆ They have a natural tendency to facilitate or to take shortcuts

# Authentication:

## Entities and deployment model

### ▷ Entities

- ♦ People
- ♦ Hosts
- ♦ Networks
- ♦ Services / servers

### ▷ Deployment model

- ♦ Along the time
  - Only when interaction starts
  - Continuously along the interaction
- ♦ Directionality
  - Unidirectional
  - Bidirectional (or mutual)

# Authentication interactions:

## Basic approaches

### ▷ Direct approach

- ◆ Provide **credentials**
- ◆ Wait for verdict
- ◆ Authenticator checks credentials against what it knows

### ▷ Challenge-response approach

- ◆ Get **challenge**
- ◆ Provide a **response** computed from the **challenge** and the **credentials**
- ◆ Wait for verdict
- ◆ Authenticator checks response for the challenge provided and the credentials it knows

# Authentication of people:

## Direct approach w/ known password

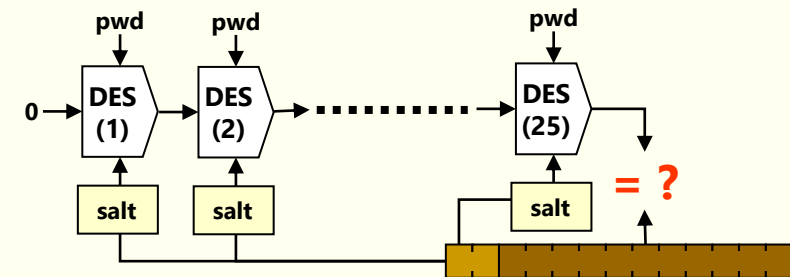
- ▷ A password is matched with a stored value
  - ♦ For a claimed identity (username)

- ▷ Personal stored value:
  - ♦ Transformed by a unidirectional function
    - Key Derivation Function (KDF)
    - Preferably slow!
    - Bcrypt, scrypt, Argon2, PBKDF2
  - ♦ UNIX: DES hash + salt
  - ♦ Linux: KDF + salt
  - ♦ Windows: digest function

$\text{DES hash} = \text{DES}_{\text{pwd}}^{25}(0)$

$\text{DES}_k^n(x) = \text{DES}_k(\text{DES}_k^{n-1}(x))$

Permutation of 12 subkeys' bit pairs with salt (12 bits)



# Authentication of people:

## Direct approach w/ known password

### ▷ Advantage

- ♦ Simplicity!
- ♦ Sharing!

### ▷ Problems

- ♦ Usage of predictable passwords
  - They enable dictionary attacks
- ♦ Different passwords for different systems
  - To prevent impersonation by malicious admins
  - But our memory has limits!
- ♦ Exchange along insecure communication channels
  - Eavesdroppers can easily learn the password
  - e.g. Unix remote services, PAP



### Top 10 2023 at Portugal by NordPass

- 1 - 123456
- 2 - 12345
- 3 - 123456789
- 4 - 12345678
- 5 - benfica
- 6 - portugal
- 7 - sporting
- 8 - 1234567890
- 9 - password
- 10 - 1234567

source: <https://nordpass.com/most-common-passwords-list/>  
Image <https://www.pinterest.com/networkboxusa/it-humor>



# Password selection jokes

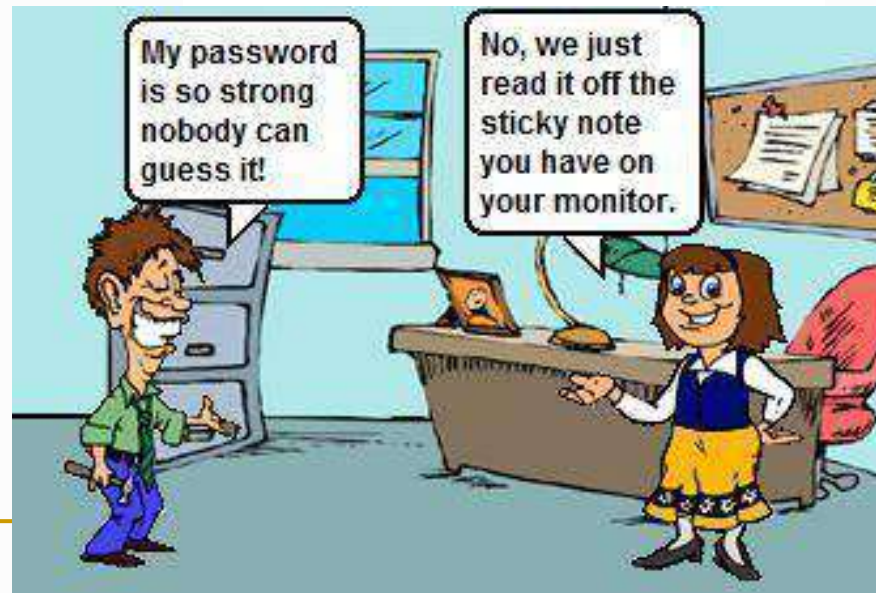
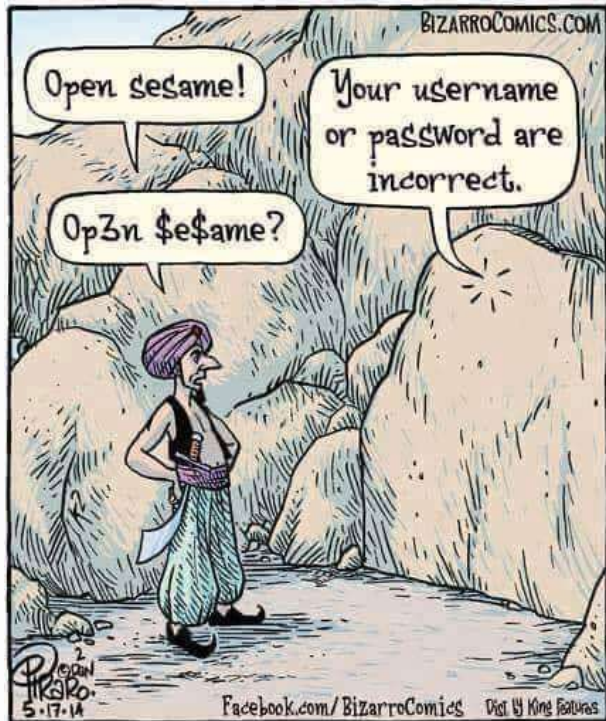


Dear IT,  
the more "secure" you try to make  
our passwords by making them  
impossible to remember, the more  
likely I am to save them all in a big  
word doc named "Passwords"

Signed,  
Everyone



Sorry, but your  
password  
must contain  
an uppercase letter,  
a number,  
a haiku,  
a gang sign,  
a hieroglyph,  
and the  
blood of a virgin.



Identification, Authentication and Authorization



# Password bloopers



# Authentication of people:

## Direct approach with biometrics

### ▷ People get authenticated using body measurements

- ♦ Biometric samples or features
- ♦ Common modalities
  - Fingerprint
  - Facial recognition
  - Palm print
  - Iris scan
  - Voice recognition
  - DNA

### ▷ Measures are compared with personal records

- ♦ Biometric references (or template)
- ♦ Registered in the system with a previous enrolment procedure





#### Infrared Camera

An infrared camera reads the dot pattern, captures an infrared image, then sends the data to the secure enclave in the A11 Bionic chip to confirm a match.

#### Flood Illuminator

Invisible infrared light helps identify your face even when it's dark.

#### Dot Projector

More than 30,000 invisible dots are projected onto your face to build your unique facial map.



# Fingerprint sensor

An optical sensor.

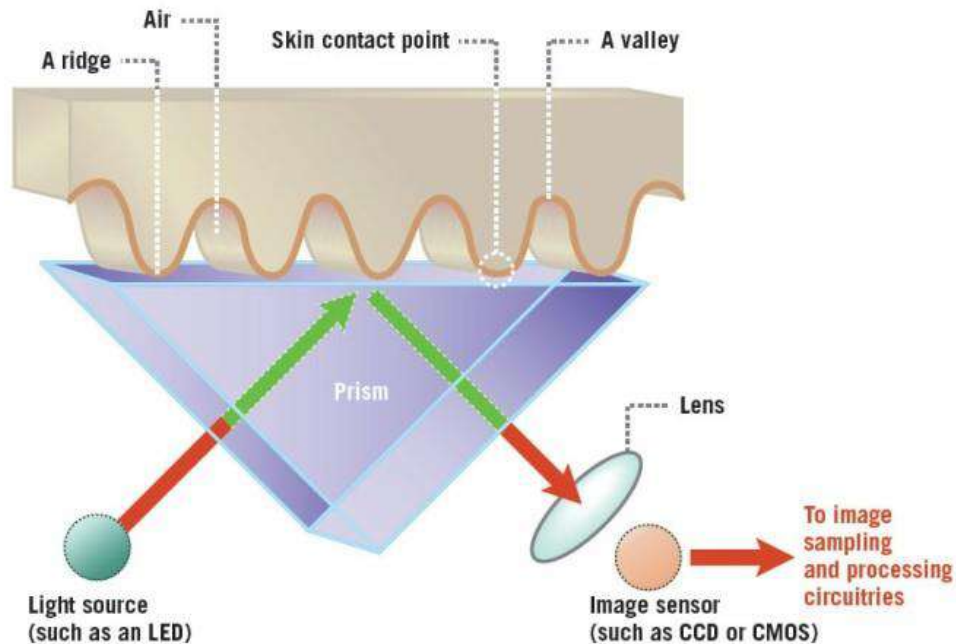
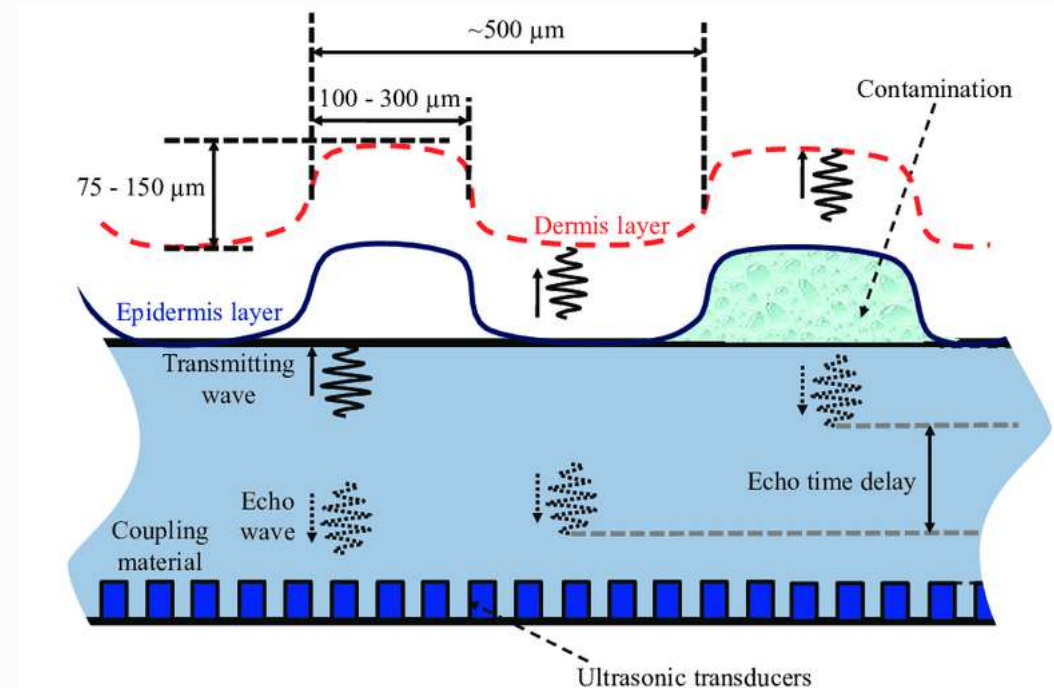


Figure 2



▷ Easy to bypass: [youtube/watch?v=hJ35ApLKpN4](https://www.youtube/watch?v=hJ35ApLKpN4)

# Biometrics: advantages

- ▷ Convenient: people do not need to use memory
  - ♦ Just be their self
- ▷ People cannot choose weak passwords
  - ♦ In fact, they don't choose anything
- ▷ Credentials cannot be transferred to others
  - ♦ One cannot delegate their own authentication
- ▷ Stealth identification
  - ♦ Interesting for security surveillance

# Biometrics: problems

- ▷ Usability
  - ◆ Comfort of people, ergonomic
  - ◆ Exploitation scenario
- ▷ Biometrics are still being improved
  - ◆ In many cases they can be easily cheated
  - ◆ Liveness detection
- ▷ People cannot change their credentials
  - ◆ Upon their robbery
- ▷ It can be risky for people
  - ◆ Removal of body parts for impersonation of the victim



Image source: <https://biometrics.mainguet.org/types/tongue.htm>

# Biometrics: problems

- ▷ Sensitivity tuning
  - ♦ Reduction of FRR (annoying)
  - ♦ Reduction of FAR (dangerous)
  - ♦ Tuning is mainly performed with the target population
    - Not with attackers!
- ▷ Not easy to deploy remotely
  - ♦ Requires trusting the remote sample acquisition system
- ▷ Can reveal personal sensitive information
  - ♦ Diseases
- ▷ Credentials cannot be (easily) copied to others
  - ♦ In case of need in exceptional circumstances

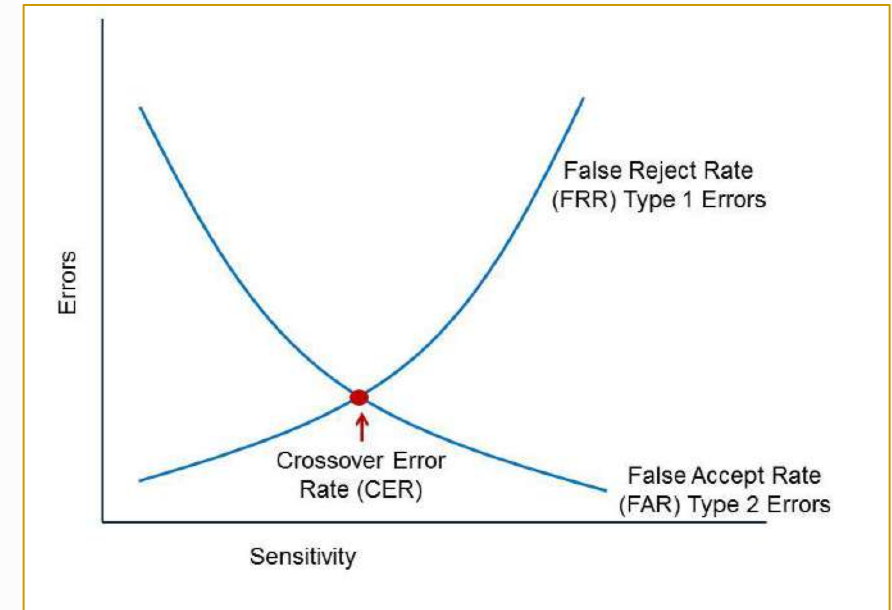


Image source: <http://www.pearsonitcertification.com/articles/article.aspx?p=1718488>

# Authentication of people:

## Direct approach with OTPs

### ▷ One-time password (OTP)

- ◆ Credential that can be used only once

### ▷ Advantage

- ◆ OTPs can be eavesdropped
- ◆ Eavesdroppers cannot impersonate the OTP owner
  - True for passive eavesdroppers
  - False for active attackers!



# Authentication of people:

## Direct approach with OTPs

### ▷ Problems

- ♦ Interactors need to know which password they should use at different occasions
  - Requires some form of synchronization
- ♦ People may need to use extra resources to maintain or generate one-time passwords
  - Paper sheets
  - Computer programs
  - Special devices, etc.

# Authentication of people:

## OTPs and secondary channels

- ▷ OTPs are codes sent through secondary channels
  - ♦ A secondary channel is a channel that is not the one where the code is going to be used
    - SMS, email, Twitter, Firebase, QR codes, NFC, etc.
  - ♦ The secondary channel provides the synchronization
    - Just-in-time provision of OTP
- ▷ Two authentications are possible
  - ♦ Confirm a secondary channel provided by a profile owner
    - In order to trust that that channel belongs to the profile owner
  - ♦ Authenticate the owner of a profile
    - Which is bound to a secondary channel

# Authentication of people:

## OTPs produced from a shared key

- ▷ HOTP (Hash-based One Time Password, RFC 4226)
  - ♦ OTP generated from a counter and a shared key
  - ♦ Counters are updated independently
- ▷ TOTP (Time-based One Time Password, RFC 6238)
  - ♦ OTP generated from a timestamp and a shared password
  - ♦ TOTP is HOTP with timestamps instead of counters
  - ♦ Clocks need a rough synchronization

# HOTP (HMAC-based one-time password, RFC 4226)

- ▷ Numeric OTP computed from shared key  $K$  and synchronized counter  $C$ 
  - ♦ Hash key and counter
    - And increase counter
  - ♦ From hash, get a (floating) portion of 31 contiguous bits
    - Dynamic Binary Code (DBC)
  - ♦ Compute a  $d$ -long decimal number
    - $d \geq 6$
  
- ▷ Issues
  - ♦ Counter synchronization upon a failure
    - If the authenticator keeps it after a failure, exhaustive search attacks are viable
    - If the authenticator always increments it, DoS attacks are possible
  - ♦ Acceptance windows
    - Mitigates minor desynchronizations, but decreases security

# TOPT (Time-based one-time password, RFC 6238)

▷ HOTP with a counter derived from time

▷  $C_T = \left\lfloor \frac{T - T_0}{T_x} \right\rfloor$

- ♦  $T$  – initial time
- ♦  $T_0$  – initial time
- ♦  $T_x$  – time interval (default: 30 seconds)

▷  $\text{TOTP}(K) = \text{HOTP}(K, C_T)$

# Token-based OTP generators: RSA SecurID



- ▶ Personal authentication token
  - ◆ Or software modules for handhelds (PDAs, smartphones, etc.)
- ▶ It generates a unique number at a fixed rate
  - ◆ Usually one per minute (or 30 seconds)
  - ◆ Bound to a person (User ID)
  - ◆ Unique number computed with:
    - A 64-bit key stored in the token
    - The actual timestamp
    - A proprietary digest algorithm (SecurID hash)
    - An extra PIN (only for some tokens)

# RSA SecurID

- ▷ OTP-based authentication
  - ♦ A user combines their User ID with the current token number
$$\text{OTP} = \text{User ID}, \text{Token Number}$$
- ▷ An RSA ACE Server does the same and checks for match
  - ♦ It also knows the person's key stored in the token
  - ♦ There must be a synchronization to tackle clock drifts
    - RSA Security Time Synchronization
- ▷ Robust against dictionary attacks
  - ♦ Keys are not selected by people

# Yubikey

## ▷ Personal Authentication Device

- ◆ USB and/or NFC

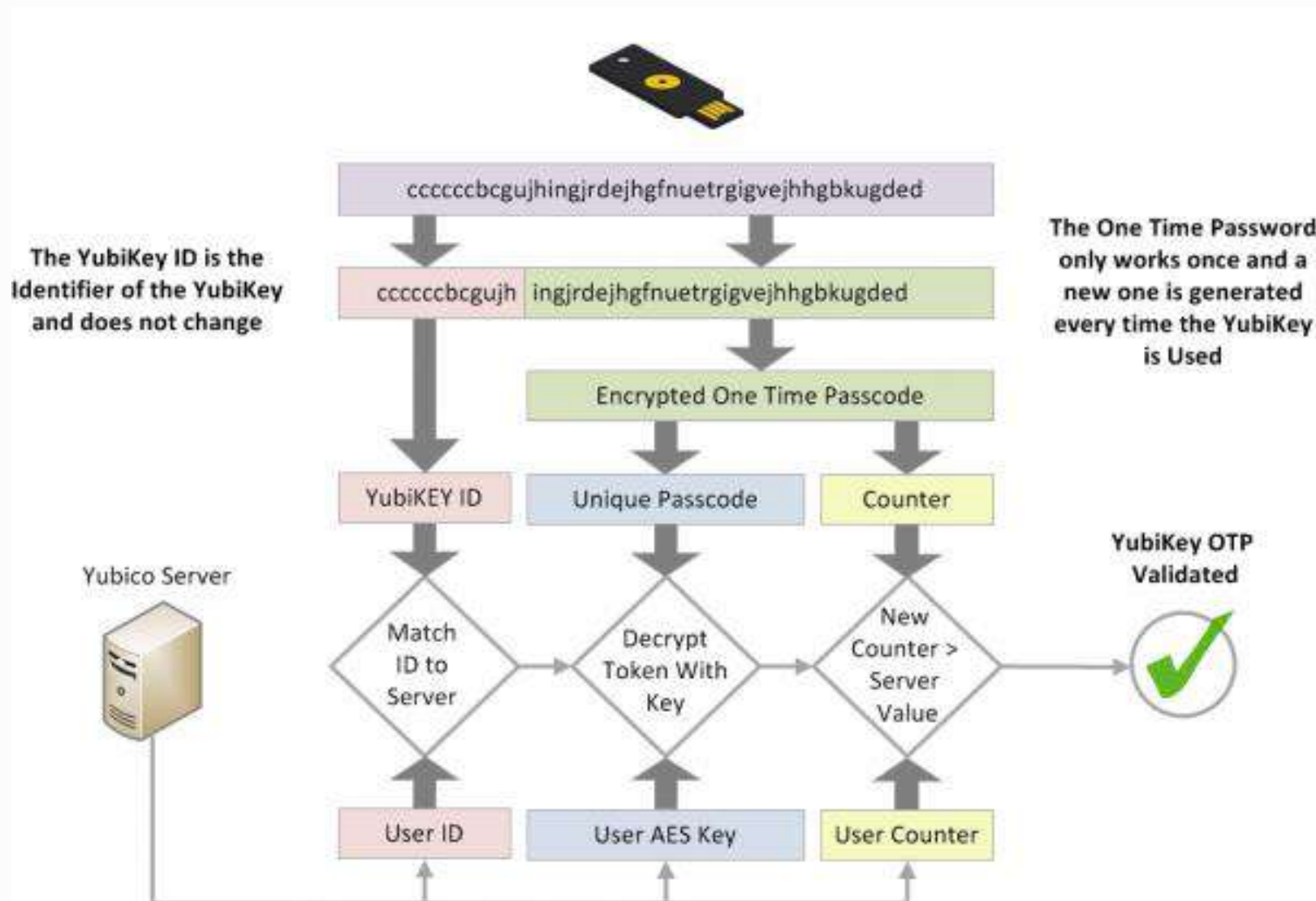
## ▷ Activation generates a 44 characters key

- ◆ Emulates a USB keyboard (besides an own API)
- ◆ Supports HOTP (events) or TOPT (Temporal)
- ◆ If a challenges is provided, user most touch the button to obtain a result
- ◆ Several algos, including AES 256



**cccjggkhcbbirdrfdnlngghhfgtrnnlgedjlfrbdeut**

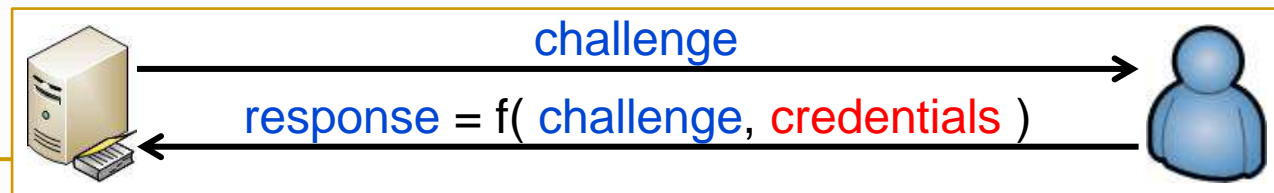




# Challenge-response approach:

## Generic description

- ▷ The authenticator provides a challenge
- ▷ The entity being authenticated transforms the challenge
  - ♦ With its authentication credentials
- ▷ The result (response) is sent to the authenticator
- ▷ The authenticator checks the response
  - ♦ Produces a similar result and checks if they match
  - ♦ Transforms the result and checks if it matches the challenge or a related value



# Challenge-response approach:

## Generic description

### ▷ Advantage

- ♦ Authentication credentials are not exposed

### ▷ Problems

- ♦ People may require means to compute responses
  - Hardware or software
- ♦ The authenticator may have to have access to shared secrets
  - How can we prevent them from using the secrets elsewhere?
- ♦ Offline dictionary attacks
  - Against recorded challenge-response dialogs
  - Can reveal secret credentials (passwords, keys)

# Challenge-response protocols: selection of challenges

- ▷ Challenges cannot be repeated for the same entity
  - ♦ Same challenge → same response
  - ♦ An active attacker can impersonate a user using a previously recorded protocol run
- ▷ Challenges should be nonces
  - ♦ Nonce: number used only once
  - ♦ Stateful services can use counters
  - ♦ Stateless services can use (large) random numbers
  - ♦ Time can be used, but with caution
    - Because one cannot repeat a timestamp

# Authentication of people:

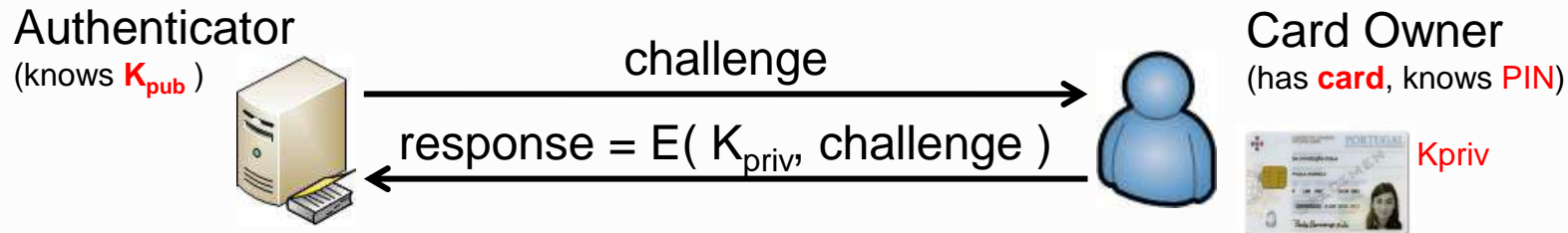
## Challenge-response with smartcards

- ▷ Authentication credentials
  - ♦ The smartcard
    - e.g. Citizen Card
  - ♦ The private key stored in the smartcard
  - ♦ The PIN to unlock the private key
- ▷ The authenticator knows
  - ♦ The corresponding public key
  - ♦ Or some personal identifier
    - Which can be related with a public key through a (verifiable) certificate



# Authentication of people:

## Challenge-response with smartcards



### ▷ Signature-based protocol

- ♦ The authenticator generates a random challenge
  - Or a value not used before
- ♦ The card owner ciphers the challenge with their private key
  - PIN-protected
- ♦ The authenticator decrypts the result with the public key
  - If the output matches the challenge, the authentication succeeds

### ▷ Encryption-based protocol

- ♦ Possible when private key decryption is available

# Authentication of people:

## Challenge-response with memorized password

### ▷ Authentication credentials

- ◆ Passwords selected by people

### ▷ The authenticator knows

- ◆ All the registered passwords; or
- ◆ A transformation of each password
  - Preferable option
  - Preferably combined with some local value (salt)
  - Preferable using a tunable function (e.g. iterations)

# Authentication of people:

## Challenge-response with memorized password

- ▷ The authenticator generates a random challenge
- ▷ The person computes a function of the challenge and password
  - ♦ e.g. a joint digest:  $\text{response} = \text{digest}(\text{challenge}, \text{password})$
  - ♦ e.g. an encryption  $\text{response} = E_{\text{password}}(\text{challenge})$
- ▷ The authenticator does the same (or the inverse)
  - ♦ If the output matches the response (or the challenge), the authentication succeeds
- ▷ Examples
  - CHAP, MS-CHAP v1/v2, S/Key



# PAP & CHAP

(RFC 1334, 1992, RFC 1994, 1996)

- ▷ Protocols used in PPP (Point-to-Point Protocol)
  - ♦ Unidirectional authentication
    - Authenticator is not authenticated
- ▷ PPP developed in 1992
  - ♦ Mostly used for dial-up connections
- ▷ PPP protocols are used by PPTP VPNs
  - ♦ e.g. vpn.ua.pt

# PAP & CHAP

(RFC 1334, 1992, RFC 1994, 1996)

## ▷ PAP (PPP Authentication Protocol)

- ♦ Simple UID/password presentation
- ♦ Insecure cleartext password transmission

## ▷ CHAP (CHallenge-response Authentication Protocol)

Aut → U: authID, challenge

U → Aut: authID, MD5( authID, pwd, challenge ), identity

Aut → U: authID, OK/not OK

- ♦ The authenticator may require a reauthentication anytime

# MS-CHAP (Microsoft CHAP)

## (RFC 2433, 1998, RFC 2759, 2000)

### ► Version 1

A → U: authID, **C**  
U → A: **R1**, **R2**  
A → U: OK/not OK

$R1 = DES_{LMPH}(C)$   
 $R2 = DES_{NTPH}(C)$

$LMPH = DEShash(pwd')$   
 $NTPH = MD4(pwd)$

$pwd' = capitalized(pwd)$

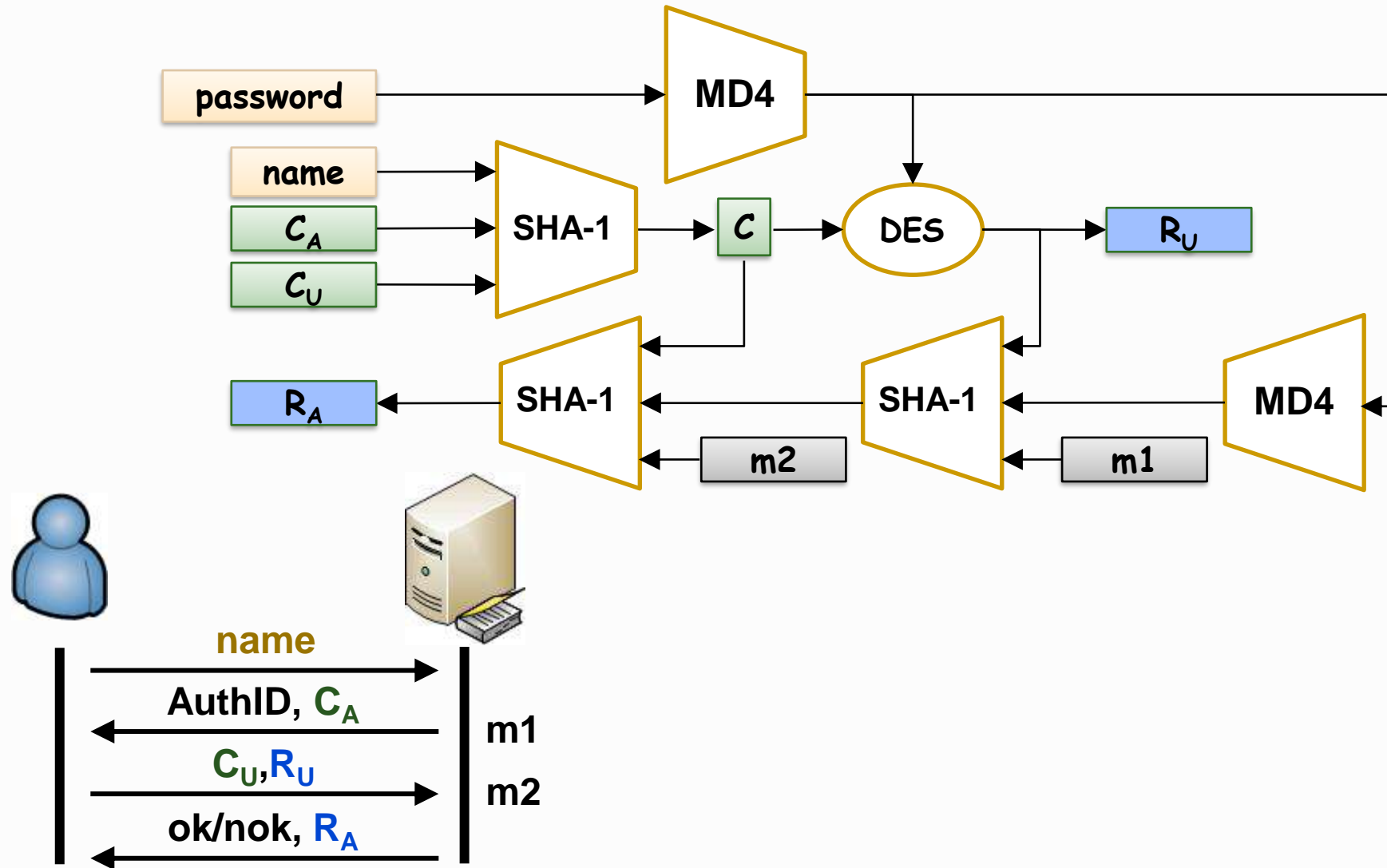
### ► Version 2

A → U: authID, **C<sub>A</sub>** ← **m1**  
U → A: **C<sub>U</sub>**, **R1** ← **m2**  
A → U: OK/not OK, **R2**

$R1 = DES_{PH}(C)$   
 $C = SHA(C_U, C_A, username)$   
 $PH = MD4(password)$   
 $R2 = SHA(SHA(MD4(PH), R1, m1), C, m2)$

- Mutual authentication
- Passwords can be updated

# MS-CHAP v2



# Authentication of people:

## Generation of OTPs with challenges

- ▷ OTPs can be produced from a challenge received
  - ♦ The fundamental protocol is password-based
    - But passwords are OTPs
  - ♦ OTPs are produced from a challenge
  - ♦ One can use several algorithms to handle OTPs

# Authentication of people:

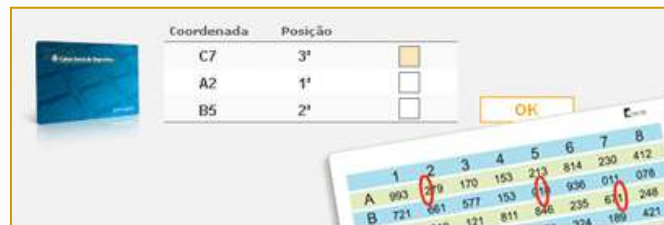
## OTPs selected from shared data

### ► Advantage:

- ♦ Shared data can be random
- ♦ No long-term short secrets to protect

### ► OTPs build from printed data

- ♦ Example: online bank codes



### ► Selection of an OTP from a printed / saved list

TPW list generated 2020-12-01 14:10 on ubuntu

```
000 jZOs ZoWc 056 Q%8x reaG 112 Pwvj ZZKE 168 b!qB ZxJ= 224 MABc m8dj
001 e6+w ZTpi 057 bUzv 67yK 113 dct6 M3m3 169 WL7T szrM 225 UxZz BgJa
002 Z5/n bWH2 058 wKps yVeW 114 gY=X X3R% 170 MFL: /3aH 226 6+w2 v2Mj
003 FOCV E=/4 059 uFUT j2Ag 115 tF6u E8mh 171 fk9H 2bvb 227 7z=8 xOPo
004 yAOF YbvS 060 NExv %hmn 116 E38O HQ85 172 Ls9u 8DXn 228 4Tow uKkz
005 SBP4 NP8r 061 P4pZ JrL9 117 qdXp FYxc 173 4ZbJ RMdL 229 G4eL od:N
006 K=Ze O7sp 062 K2ys +Wvb 118 wvcC =85E 174 zPnj 2rA8 230 54f= P5xH
007 9Bqp 9E97 063 yBP= rE39 119 sIdC mqDi 175 5Kek oaI+ 231 2mwJ uAJV
008 mhmP BjMX 064 Yayh Y=uM 120 3bcx 4cSB 176 Z:Q6 dsCn 232 9kB: :xap
009 3KLK w2ck 065 t6FW er=a 121 dE:9 97vL 177 :aFX buOX 233 eR3O mben
010 %MXc 3x7: 066 J+y: DpVR 122 BWMz 65GX 178 ZTFQ IJHS 234 D+rQ WPwO
011 jRTZ Ftsu 067 wERC JF5Q 123 XaYp XUFN 179 A3tN 9p5v 235 o2m% PmS=
012 :UtJ xzLb 068 bCuN eGIX 124 =AWj %J%p 180 MCre BFgp 236 dz3+ EZRd
013 PgVK M+vS 069 mo+d taKT 125 wODU =mwS 181 56IO VhB6 237 3RNm xFHY
014 NT/s F8mf 070 XiSO rNDz 126 qCnE 5Kp/ 182 6=im Rht4 238 rLKG N:3c
015 HJ+Z nmCK 071 UA8U qECw 127 pafc foFR 183 RM7Q 9Twe 239 d4z2 HTuc
016 =OKR rInR 072 PjGj n:oS 128 DSdk ufcM 184 eKmq avXK 240 PnPM NThB
017 hKhD UrZX 073 XYTD ZKfn 129 dFAC /zVX 185 wEtS X3P5 241 gTiF 69k4
018 tCbN wfiR 074 q4B2 uPh9 130 Ix2: XsGF 186 RgQe OnoQ 242 sy/U bpZJ
019 8R2V MNx5 075 =qv2 oMr9 131 gc2Y YSWd 187 WAF9 5Ac/ 243 =vS5 pXxx
020 9wC7 Q8Vh 076 E/2P e5I= 132 GhI6 P4bP 188 u/K4 MyTy 244 PLKS roct
021 q2Pg qTuF 077 TaFC /cs7 133 b=aP UeSQ 189 3FOC /9nd 245 ACj4 4s:A
022 9uxI P:dy 078 ILZ6 Tvpi 134 wLiT AgSS 190 vhty RMe2 246 GmvO Cp2N
023 qtUo GAX3 079 bD2x GRet 135 tDoH 7qXH 191 oHnH y5KG 247 /mcs Gair
024 ZXfQ 8HAV 080 W%Ig c:T= 136 VgNd Evz6 192 CT6a HrAc 248 yodz vKtf
025 E8/N 7kxE 081 eRwK nzx8 137 =Gsw EHWg 193 %Xm/ =pbj 249 z:BY BW5L
026 guVQ Ljhb 082 xfeT z9ta 138 9OOX RtQT 194 f4gE NHOB 250 %zBj jsvd
027 Y%8n :y6f 083 a3ob rjvf 139 KfMj VE6k 195 %38V 3ZEh 251 %ZNR vDBY
028 rDmL X9pH 084 q89w pFka 140 Ym+ hmaY 196 =Tho ORDr 252 QAot tbe7
029 awpG M6Rt 085 7/no X/m3 141 gGub j:hZ 197 Ppjo AwgM 253 c3F4 Q4oD
030 Jt:w DGUX 086 5iWg L=Am 142 4iBd IpBy 198 %mdT opDB 254 hwQw BQoO
031 aFh2 uPSP 087 cMxK 6tjU 143 dsRp NTN/ 199 rtm2 OGN+ 255 fxeC Q%mb
032 mx4E GIVc 088 guQR h+Kd 144 Mk6X S/qJ 200 LHQo rOVN 256 o5zt +xMm
033 xh7V CYgj 089 Viw/ AaFq 145 Otm3 %NfF 201 IfsK JgkK 257 OTKn Co4p
034 ZjBZ xW:j 090 bDcc mMUb 146 YR28 OxTH 202 d5zO tWbb 258 ceb+ s=2%
035 RwHa wcV: 091 KuOk nF/G 147 8AKW 99u7 203 dMOJ d:/I 259 moQh RoOK
036 XRsS NPGR 092 PSY3 expc 148 onxi /gBe 204 L3WG AH8K 260 SMuY 9ArI
037 B6VS EKT/ 093 TiRy 5ZQj 149 ozwu O2br 205 EzB6 =Uc9 261 AGPD pYdt
038 %J6/ 6HR7 094 /HQz jeUC 150 NVR7 tsgm 206 46f9 g9BX 262 i%7h RcnF
039 eYQa fwAg 095 Rzd2 /3kf 151 Ewer s9/f 207 Va2M Lr=S 263 dn8j 955y
040 Wt4 3TQA 096 XjtZ ARep 152 bxc5 /+Sv 208 :+md drVx 264 tjYf uP22
041 4Pwp W68U 097 kZBy t8WW 153 zfgv Xz8q 209 ayGk e8oF 265 Q%WZ BcR=
042 Myi3 j9k9 098 b:Kx PuI6 154 Zsp2 UaNN 210 9hv6 Vu3: 266 XC%Q GpOG
043 QH58 kQhy 099 z3w7 B8Qa 155 PPOg ih8e 211 kv8n kn+o 267 /:IK Ik3w
044 zYCi y5NZ 100 4wdV =ak 156 bhy4 UkfN 212 qL8a cDz= 268 2ei5 dY43
045 s=Dz AF2F 101 BITZ JP9Q 157 Ufv% T:Wx 213 LTWv 96a9 269 ABM 3mN2
046 7AFp Rctz 102 4WPg HNko 158 XsED ywk+ 214 ywp+ Xq2P 270 7yoU fB6w
047 ao8H 5PHh 103 g6mJ T3YK 159 mTuL ZsrQ 215 ST2: qzCF 271 uA:4 Q+bJ
048 /vgM /h%c 104 wsiw x3/U 160 nHGY aIB+ 216 CnOH Wf6P 272 =aJb w97Z
049 TznT mhbl 105 UTNP 6vjE 161 :Cin i7:4 217 IFJh x5cZ 273 5dSN evT=
050 6Tha :rNG 106 zKAu 8Qt6 162 /ECm Z6yy 218 8U5W Xu% 274 eim+ eWz%
051 DM9G wb37 107 tojS KwgB 163 PbA7 3rja 219 N=8= pCiU 275 +Qm% %zZo
052 Nuze JzMn 108 66n8 jhKk 164 O:I7 :2qO 220 CPuy =y3K 276 Qs4k i5HP
053 RfE% 7:BN 109 Pt9u +/p+ 165 =vuu O:dt 221 O5vF XaoK 277 +2KS +rvR
054 mJqq hmK= 110 Aped PQbv 166 9KAR Fazz 222 umBy =t5d 278 u47 XSUO
055 FI/d nFk= 111 PF+k FdFZ 167 x%a= 222P 223 iC2I =2eJ 279 xP6i 3TCQ
```

# S/Key (RFC 2289, 1998)

## ▷ Authentication credentials

- ♦ A password (**pwd**)

## ▷ The authenticator knows

- ♦ The last used one-time password (**OTP**)
- ♦ The last used OTP **index**
  - Defines an order among consecutive OTPs
- ♦ A **seed** value for each person's OTPs
  - The seed is similar to a UNIX salt

# S/Key setup

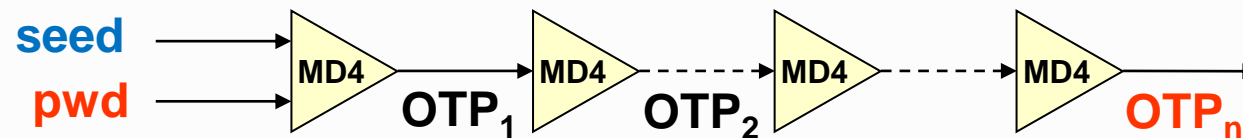
▷ The authenticator defines a random seed

▷ The person generates an initial OTP as:

$$\text{OTP}_n = h^n(\text{seed}, \text{pwd}), \text{ where } h = \text{MD4}$$

♦ Some S/Key versions also use MD5 or SHA-1

▷ The authenticator stores seed, n and  $\text{OTP}_n$  as authentication credentials





# S/Key authentication protocol

- ▷ Authenticator sends seed & index of the person
  - ♦ They act as a challenge
- ▷ The person generates index-1 OTPs in a row
  - ♦ And selects the last one as result
  - ♦  $\text{result} = \text{OPT}_{\text{index}-1}$
- ▷ The authenticator computes  $h(\text{result})$  and compares the result with the stored  $\text{OPT}_{\text{index}}$ 
  - ♦ If they match, the authentication succeeds
  - ♦ Upon success, stores the recently used index & OTP
    - index-1 and  $\text{OPT}_{\text{index}-1}$

# S/Key

## ▷ Advantages

- ◆ Users passwords are unknown to authenticators
- ◆ OTPs can be used as ordinary passwords

## ▷ Disadvantages

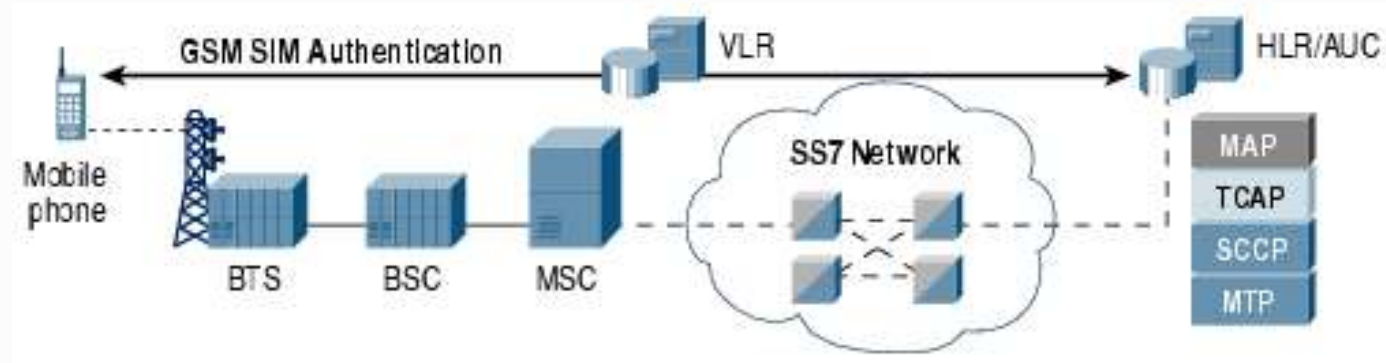
- ◆ People need an application to compute OTPs
- ◆ Passwords can be derived using dictionary attacks
  - From data stored in authenticators
  - From captured protocol runs

# Authentication of people:

## Challenge-response with shared key

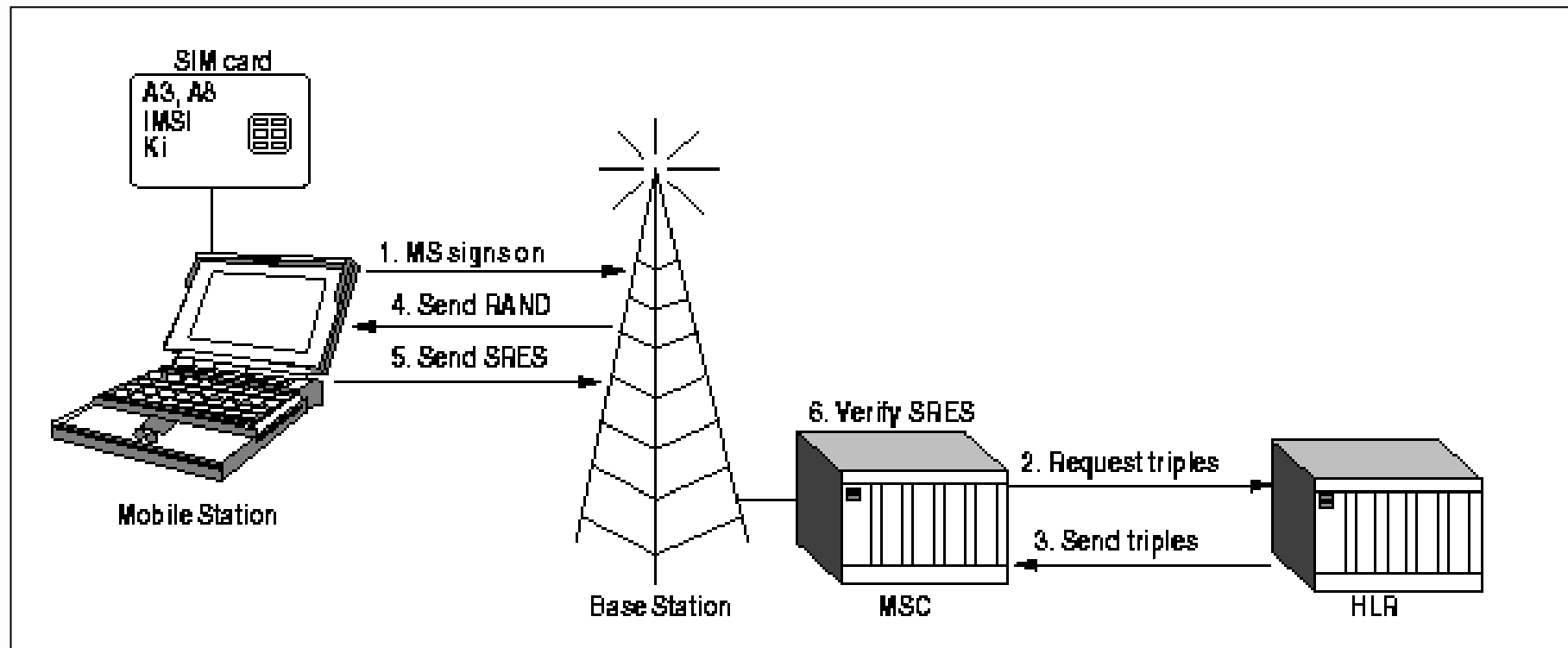
- ▶ Uses a shared key instead of a password
  - ◆ Robust against dictionary attacks
  - ◆ Requires some token to store the key
- ▶ Example:
  - ◆ GSM

# GSM: authentication architecture



- ▷ Based on a secret key shared between the HLR and the station
  - ♦ 128 Ki, stored in the station's SIM card
  - ♦ Can only be used after entering a PIN
- ▷ Algorithms (initially not public):
  - ♦ A3 for authentication
  - ♦ A8 for generating a session key
  - ♦ A5 for encrypting the communication
- ▷ A3 and A8 implemented by SIM card
  - ♦ Can be freely selected by the operator

# GSM: mobile station authentication



# GSM: mobile station authentication

- ▷ MSC fetches trio from HLR
  - ♦  $RAND, SRES, K_c$
  - ♦ In fact more than one are requested
- ▷ HLR generates  $RAND$  and corresponding trio using subscriber's  $K_i$ 
  - ♦  $RAND$ , random value (128 bits)
  - ♦  $SRES = A3(K_i, RAND)$  (32 bits)
  - ♦  $K_c = A8(K_i, RAND)$  (64 bits)
- ▷ Usually operators use COMP128 for A3/A8
  - ♦ Recommended by the GSM Consortium
  - ♦  $[SRES, K_c] = COMP128(K_i, RAND)$

# Host authentication

- ▷ By name or address
  - ◆ DNS name, IP address, MAC address, other
  - ◆ Extremely weak, no cryptographic proofs
    - Nevertheless, used by many services
    - e.g. NFS, TCP *wrappers*
- ▷ With cryptographic keys
  - ◆ Keys shared among peers
    - With an history of usual interaction
  - ◆ Per-host asymmetric key pair
    - Pre-shared public keys with usual peers
    - Certified public keys with any peer

# Service / server authentication

## ▷ Host authentication

- ♦ All co-located services/servers are indirectly authenticated

## ▷ Per-service/server credentials

- ♦ Shared keys
  - When related with the authentication of people
  - The key shared with each person can be used to authenticate the service to that person
- ♦ Per-service/server asymmetric key pair
  - Certified or not



# TLS (Transport Layer Security, RFC 8446)

- ▷ Secure communication protocol over TCP/IP
  - ♦ Created upon SSL V3 (Secure Sockets Layer)
  - ♦ Manages per-application secure sessions over TCP/IP
    - Initially conceived for HTTP traffic
    - Actually used for other traffic types
- ▷ There is a similar version for UDP (DTLS, RFC 6347)
- ▷ Security mechanisms
  - ♦ Communication confidentiality and integrity
    - Key distribution
  - ♦ Authentication of communication endpoints
    - Servers (or, more frequently, services)
    - Client users
    - Both with asymmetric key pairs, typically with certified public keys

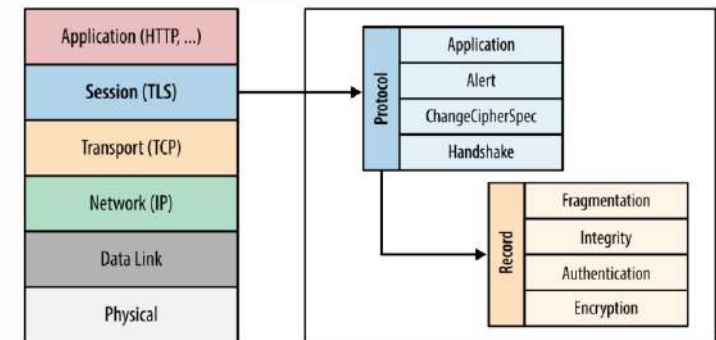
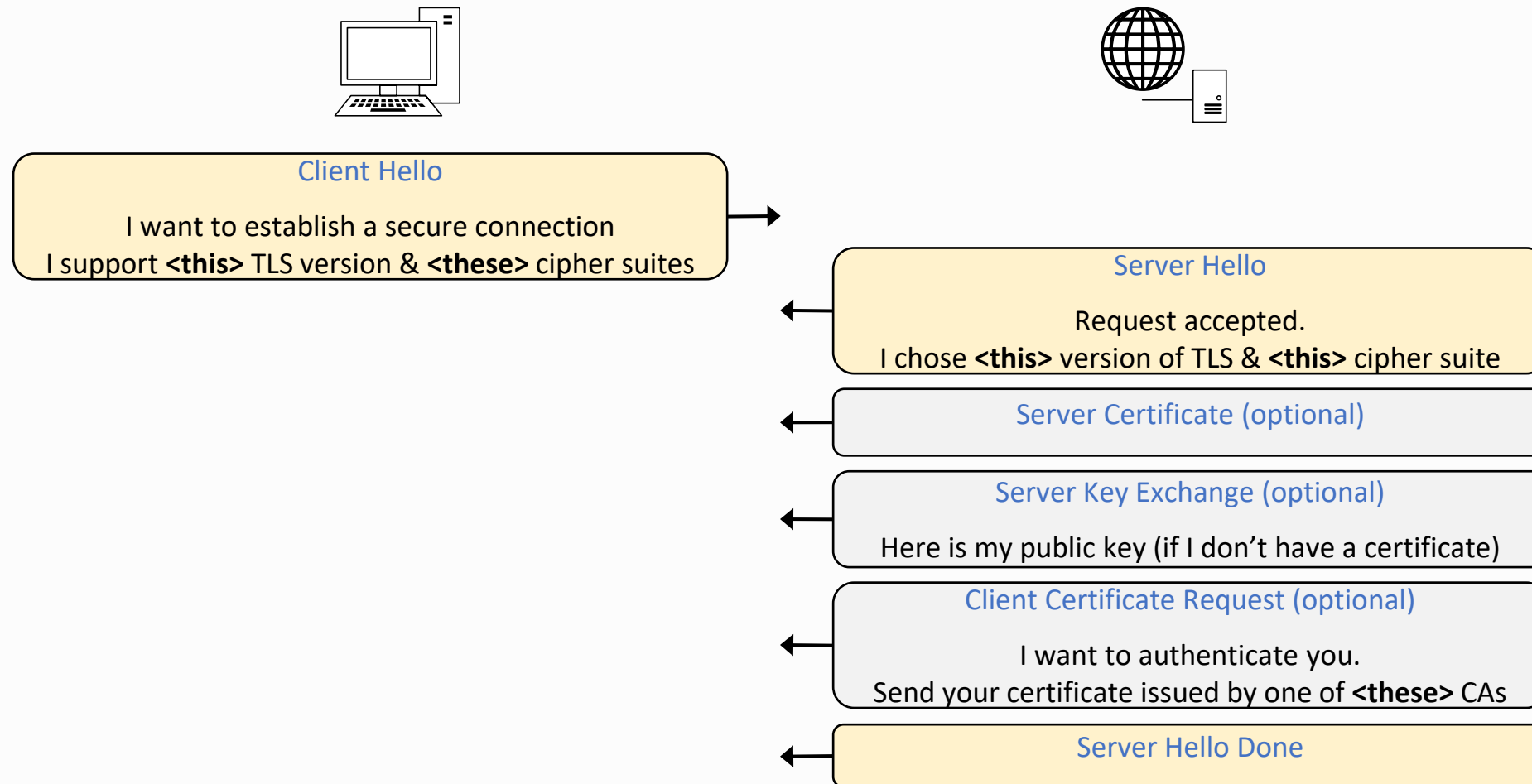
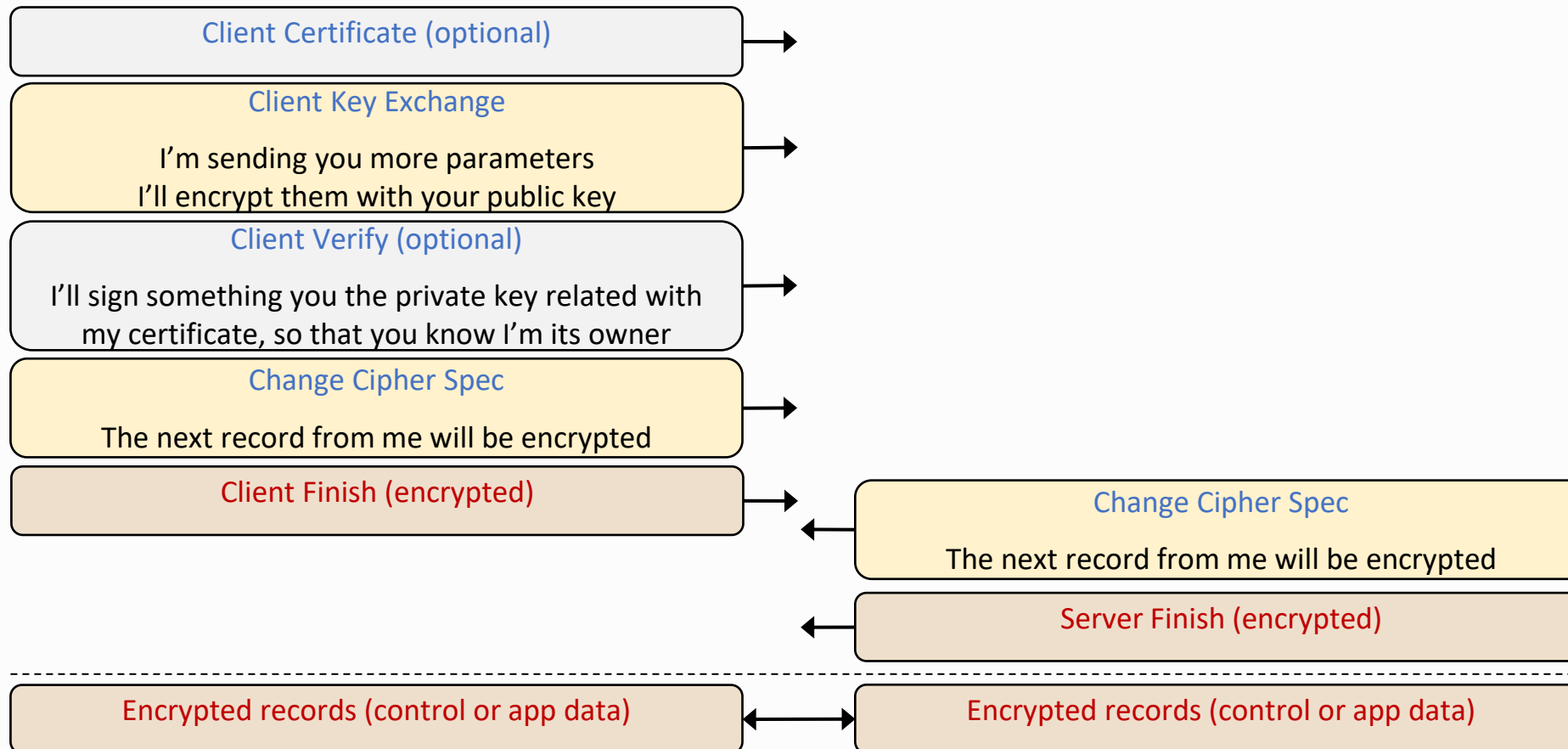
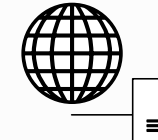
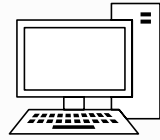


Image source: <https://hpbm.co/transport-layer-security-tls/>

# TLS interaction diagrams (1<sup>st</sup> part)



# TLS interaction diagrams (2<sup>nd</sup> part)



# TLS Ciphersuites

- ▷ If a server supports a single algorithm, it not expected for all clients to also support it
  - ♦ More powerful/limited, older/newer
- ▷ The Ciphersuite concept allows the negotiation of mechanisms between client and server
  - ♦ Both send their supported ciphersuites, and select one they both share
  - ♦ TLS v1.3: O servidor escolhe
- ▷ Example: ECDHE-RSA-AES128-GCM-SHA256
- ▷ Format:
  - ♦ Key negotiation algorithm: ECDHE
  - ♦ Authentication algorithm: RSA
  - ♦ Cifra algorithm, and cipher mode: AES-128 GCM
  - ♦ Integrity control algorithm: SHA256

# SSH (Secure Shell, RFC 4251)

## ▷ Alternative to telnet/rlogin protocols/applications

- ♦ Manages secure consoles over TCP/IP
- ♦ Initially conceived to replace telnet
- ♦ Actually used for other applications
  - Secure execution of remote commands (rsh/rexec)
  - Secure copy of contents between machines (rcp)
  - Secure FTP (sftp)
  - Creation of arbitrary secure tunnels (inbound/outbound/dynamic)

## ▷ Security mechanisms

- ♦ Communication confidentiality and integrity
  - Key distribution
- ♦ Authentication of communication endpoints
  - Servers / machines
  - Client users
  - Both with different techniques

# SSH authentication mechanisms

## ▷ Server: with asymmetric keys pair

- ♦ Inline public key distribution
  - Not certified!
- ♦ Clients cache previously used public keys
  - Caching should occur in a trustworthy environment
  - Update of a server's key raises a problem to its usual clients

## ▷ Client users: configurable

- ♦ Username + password
  - By default
- ♦ Username + private key
  - Upload of public key in advance to the server

# Single Sign-On (SSO)

- ▷ Unique, centralized authentication for a set of federated services
  - ♦ The identity of a client, upon authentication, is given to all federated services
  - ♦ The identity attributes given to each service may vary
  - ♦ The authenticator is called **Identity Provider (IdP)**
- ▷ Examples
  - ♦ SSO authentication @ UA
    - Performed by a central IdP (idp.ua.pt)
    - The identity attributes are securely conveyed to the service accessed by the user

# Authentication metaprotocols

- ▷ Generic authentication protocols that encapsulate other authentication protocols
- ▷ Examples
  - ◆ EAP (Extensible Authentication Protocol)
    - Used in 802.1X (WiFi, enterprise mode)
    - e.g. PEAP (Protected EAP) and EAP-TLS run over EAP
  - ◆ ISAKMP(Internet Security Association and Key Management Protocol)
    - Formerly used in IPSec
    - e.g. IKE v1 (Internet Key Exchange) runs over ISAKMP



# Authentication services

- ▶ Trusted third parties (TTP) used for authentication
  - ◆ But often combined with other related functionalities
- ▶ AAA services
  - ◆ Authentication, Authorization and Accounting
  - ◆ e.g. RADIUS

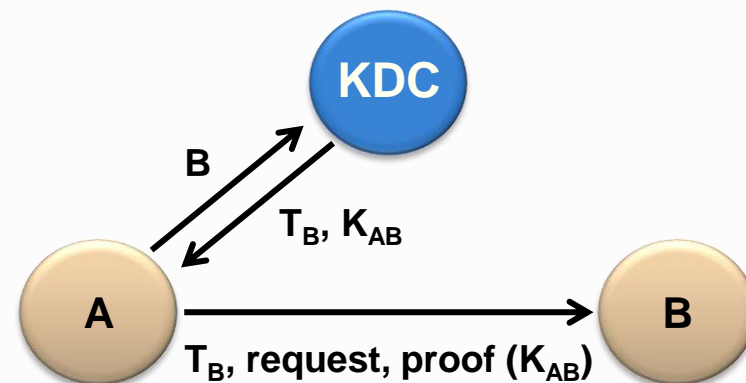
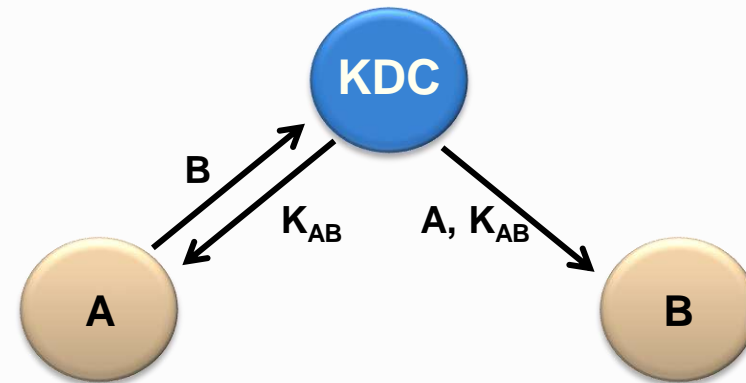
# Key distribution services

## ▷ Services that distribute a shared key for authenticated entities

- ◆ That key can then be used by those entities to protect their communication and ensure source authentication

## ▷ Examples

- ◆ 802.1X (Wi-Fi, enterprise mode)
- ◆ Kerberos



---

# PAM

## (Pluggable Authentication Modules)

---

# Motivation

## ▷ Users

- ◆ Unification of authentication mechanisms for different applications

## ▷ Manufacturers

- ◆ Authenticated access to services independent of authentication mechanisms

## ▷ Administrators

- ◆ Easy orchestration of authentication mechanisms different services requiring client authentication
- ◆ Flexibility to configure specific authentication mechanisms for each host

## ▷ Manufacturers and Administrators

- ◆ Flexible and modular approach for integrating novel authentication mechanisms

# PAM: features

- ▷ Independent authentication protocols / mechanisms
  - ♦ Linux password, S/Key, smartcards, biometrics, etc.
  - ♦ One module per protocol / mechanism
- ▷ Orchestration of protocols / mechanisms
  - ♦ Alone or combined
  - ♦ AND and OR combinations
  - ♦ Application-independent
- ▷ Several interface approaches
  - ♦ Input from text consoles of graphical windows
  - ♦ Access to special devices (smart-cards, biometric readers, etc.)

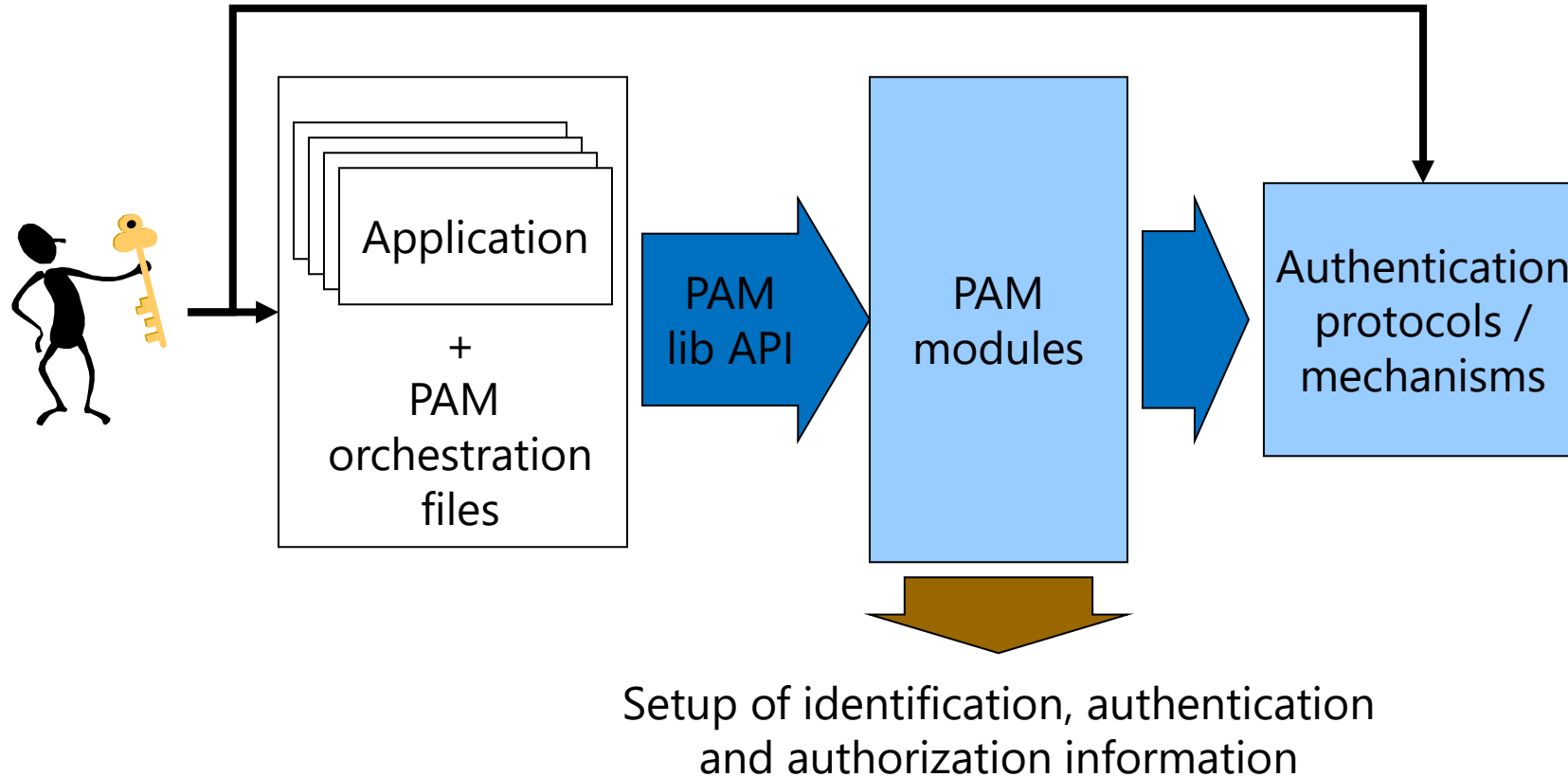
# PAM: features

- ▷ Modular and extensible architecture
  - ◆ Dynamic loading of required modules
  - ◆ Handling of several actions besides authentication
    - Password management
    - Accounting management
    - Session management
- ▷ Default orchestration per host
  - ◆ Defined by the administrator
    - Username/password, biometrics, smart-cards, etc.
- ▷ Application-specific orchestrations
  - ◆ Each application can use a unique orchestration

# Classic Unix authentication

- ▷ Requested input: username + password
- ▷ Validation
  - ◆ Active account for username
    - Entry with the username in the /etc/passwd file
  - ◆ Transformed password for that username
    - Entry with the username in the /etc/shadow file
  - ◆ Transformation of the provided password with the function and the salt used for that username
  - ◆ Comparison with the stored transformation
- ▷ Obtained credentials
  - ◆ UID + GID [+ list of secondary GIDs]
  - ◆ New process descriptor (login shell)

# PAM: Architecture





# PAM: Actions

- ▷ Authentication (**auth**)
  - ♦ Identity verification
- ▷ Account Management (**account**)
  - ♦ Enforcement of access policies based on account properties
- ▷ Password Management (**password**)
  - ♦ Management of authentication credentials
- ▷ Session Management (**session**)
  - ♦ Verification of operational parameters
  - ♦ Setup of session parameters
    - max memory, max file descriptions, graphical interface configuration, ...

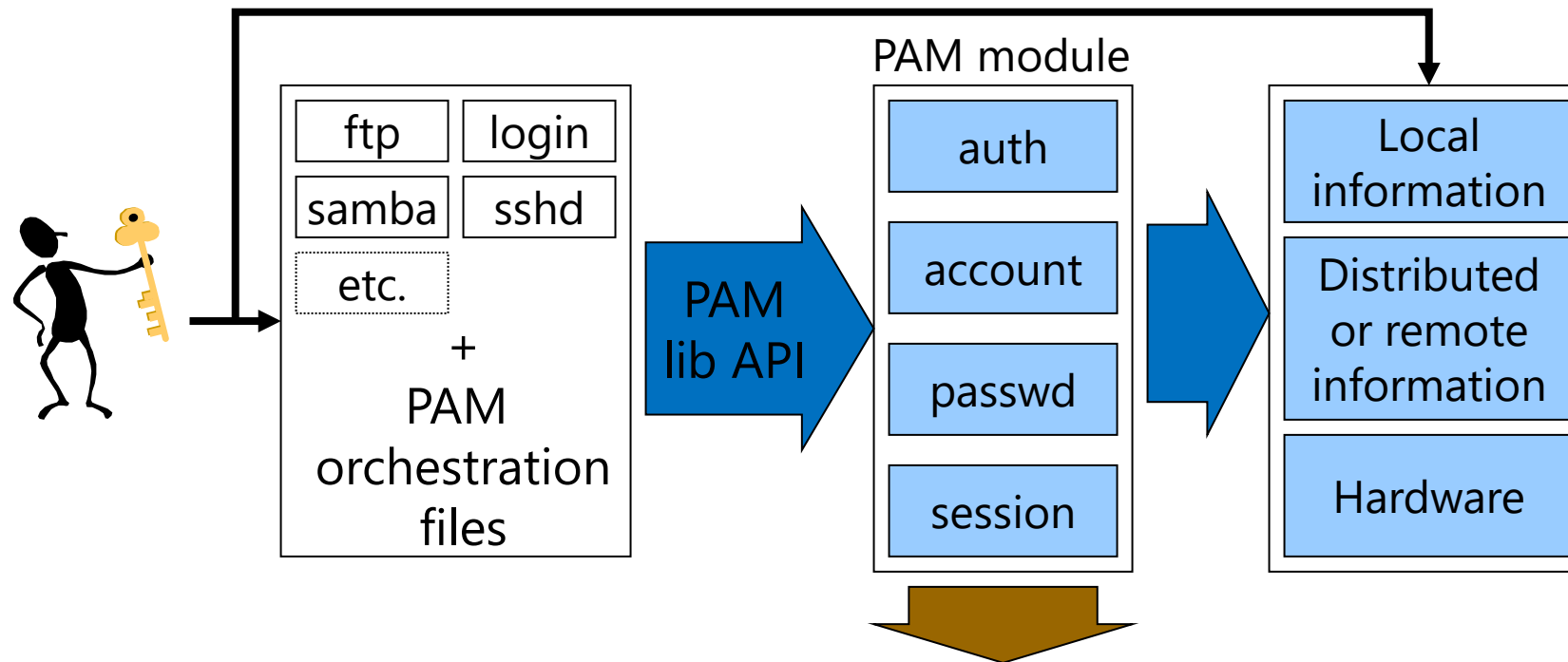
# PAM: Modules

- ▷ Dynamically loaded (*shared libraries*)
  - ♦ `/lib/security/pam_*.so`
  - ♦ `/lib/x86_64-linux-gnu/security/pam_*.so`
- ▷ Standard API
  - ♦ Functions provided by the modules that are used
    - C interfaces
    - Python wrapper exists
  - ♦ Decision provided on returned code
    - `PAM_SUCCESS`
    - `PAM_AUTH_ERR`, `PAM_AUTHINFO_UNAVAIL`, etc...
  - ♦ Not all functions need to be implemented
    - A module does not need to implement all 4 actions

# PAM: orchestration files

- ▷ Typically, one per PAM client application
  - ♦ e.g. `/etc/pam.d/ftp` or `/etc/pam.d/ssh`
  - ♦ Can use shared files: `/etc/pam.d/common-auth`
- ▷ Specify how the actions should be applied
  - ♦ Their mechanisms (modules)
  - ♦ Their parameters
  - ♦ Their termination, with or without success
- ▷ Each module uses a particular set of resources
  - ♦ Local files
    - `/etc/passwd`, `/etc/shadow`, `/etc/groups`, etc.
  - ♦ Distributed information or located in remote servers
    - NIS, Kerberos, LDAP, etc.

# PAM: Detailed Architecture



Setup of identification, authentication and authorization information

# PAM APIs:

## PAM lib (used by applications)

- ▷ Start/end of the PAM lib

`pam_start( service, user name, callback, &pam_handle )`  
`pam_end( pam_handle, status )`

- ▷ Module specific data

`pam_get_data()`  
`pam_set_data()`

`pam_get_item()`  
`pam_set_item()`

- ▷ "auth" action

`pam_authenticate( pam_handle, flags )`  
`pam_setcred( pam_handle, flags )`

- ▷ "account" action

`pam_acct_mgmt( pam_handle, flags )`

- ▷ "passwd" action

`pam_chauthtok( pam_handle, flags )`

- ▷ "session" action

`pam_open_session( pam_handle, flags )`  
`pam_close_session( pam_handle, flags )`

# Orchestration of PAM actions

- ▷ Sequence of module invocations per action
  - ♦ By default, modules are executed sequentially
  - ♦ Each module has its own parameters and calling semantic
    - Required, requisite, sufficient, optional
    - [...]
  - ♦ Execution proceeds until the end, or failure
    - To better hide the source of a failure, module execution can either abort immediately or delay the failure upon executing the entire sequence
  - ♦ Applications can recover from failures

# PAM APIs:

## PAM modules' API

- ▷ "auth" action

- `pam_sm_authenticate( pam_handle, flags )`
  - `pam_sm_setcred( pam_handle, flags )`

- ▷ "account" action

- `pam_sm_acct_mgmt( pam_handle, flags )`

- ▷ "passwd" action

- `pam_sm_chauthtok( pam_handle, flags )`

- ▷ "session" action

- `pam_sm_open_session( pam_handle, flags )`
  - `pam_sm_close_session( pam_handle, flags )`

# PAM: Module invocation

- ▷ Syntax: **action control module [parameters]**
- ▷ Control is specified for each action and module
  - requisite**
  - required**
  - sufficient**
  - optional**

**[success=ok/number default=ignore/die/bad ...]**



# PAM: Module invocation

- ▷ **required:** The module result must be successful for authentication to continue.
  - ♦ If the test fails at this point, the user is not notified until the results of all module tests that reference that interface are complete.
- ▷ **requisite:** The module result must be successful for authentication to continue.
  - ♦ However, if a test fails at this point, the user is notified immediately with a message reflecting the first failed required or requisite module test.
- ▷ **sufficient:** The module result is ignored if it fails.
  - ♦ If the result of a module flagged sufficient is successful and no previous modules flagged required have failed, then no other results are required and the user is authenticated

# PAM: Module invocation

- ▷ **optional**: The module result is ignored.
  - ♦ A module flagged as optional only becomes necessary for successful authentication when no other modules reference the interface.
- ▷ **include**: Unlike the other controls, this does not relate to how the module result is handled.
  - ♦ This flag pulls in all lines in the configuration file which match the given parameter and appends them as an argument to the module.

# Configuration files: /etc/pam.d/login

```
auth optional pam_faildelay.so delay=3000000
auth [success=ok new_authtok_reqd=ok ignore=ignore user_unknown=bad default=die] pam_securetty.so
auth requisite pam_nologin.so
```

```
session [success=ok ignore=ignore module_unknown=ignore default=bad] pam_selinux.so close
session required pam_loginuid.so
session [success=ok ignore=ignore module_unknown=ignore default=bad] pam_selinux.so open
session required pam_env.so readenv=1
session required pam_env.so readenv=1 envfile=/etc/default/locale
```

```
@include common-auth
auth optional pam_group.so
```

```
session required pam_limits.so
session optional pam_lastlog.so
session optional pam_motd.so motd=/run/motd.dynamic
session optional pam_motd.so noudate
session optional pam_mail.so standard
session optional pam_keyinit.so force revoke
```

```
@include common-account
@include common-session
@include common-password
```

# PAM orchestration files:

## Advanced decision syntax

▷ [value=action value=action ...]

▷ Actions:

- ♦ **ignore**: take no decision
- ♦ **bad**: continue, but the final decision will be a **failure**
- ♦ **die**: terminate immediately with **failure**
- ♦ **ok**: continue, so far the decision is **success**
- ♦ **done**: terminate immediately with **success**
- ♦ **reset**: clear the entire state and continue
- ♦ **N** (unsigned integer): same as ok + jump over **N** lines

# PAM orchestration files:

## Advanced decision syntax

### ▷ Values (return codes)

- ♦ *success*
- ♦ *open\_err*
- ♦ *symbol\_err*
- ♦ *service\_err*
- ♦ *system\_err*
- ♦ *buf\_err*
- ♦ *perm\_denied*
- ♦ *auth\_err*
- ♦ *cred\_insufficient*
- ♦ *authinfo\_unavail*
- ♦ *user\_unknown*
- ♦ *maxtries*
- ♦ *new\_authtok\_reqd*
- ♦ *acct\_expired*
- ♦ *session\_err*
- ♦ *cred\_unavail*
- ♦ *cred\_expired*
- ♦ *cred\_err*
- ♦ *no\_module\_data*
- ♦ *conv\_err*
- ♦ *authtok\_err*
- ♦ *authtok\_recover\_err*
- ♦ *authtok\_lock\_busy*
- ♦ *authtok\_disable\_aging*
- ♦ *try\_again*
- ♦ *ignore*
- ♦ *abort*
- ♦ *authtok\_expired*
- ♦ *module\_unknown*
- ♦ *bad\_item*
- ♦ *conv\_again*
- ♦ *incomplete*
- ♦ *default*
  - *Any not specified*

# PAM orchestration files:

## Simplified decision syntax

### ▷ High-level decisions definitions

- ♦ **requisite**
  - [success=ok new\_authtok\_reqd=ok ignore=ignore default=die]
- ♦ **required**
  - [success=ok new\_authtok\_reqd=ok ignore=ignore default=bad]
- ♦ **sufficient**
  - [success=done new\_authtok\_reqd=ok default=ignore]
- ♦ **optional**
  - [success=ok new\_authtok\_reqd=ok default=ignore]

# Scenario 2 – LDAP auth with local backoff

```
auth      required    /lib/security/pam_env.so
auth      sufficient  /lib/security/pam_unix.so likeauth nullok
auth      sufficient  /lib/security/pam_ldap.so use_first_pass
auth      required    /lib/security/pam_deny.so

account   required    /lib/security/pam_unix.so
account   [default=bad success=ok user_unknown=ignore] pam_ldap.so

password  required    /lib/security/pam_cracklib.so retry=3
password  sufficient  /lib/security/pam_unix.so nullok use_authok md5 shadow
password  sufficient  /lib/security/pam_ldap.so
password  required    /lib/security/pam_deny.so

session   required    /lib/security/pam_limits.so
session   required    /lib/security/pam_unix.so
session   optional    /lib/security/pam_ldap.so
```

**Try local accounts (for offline admin) and then LDAP**

**Allow changing the password of both local and remote accounts**

# Scenario 1 – Local authentication

auth	required	/lib/security/pam_env.so
auth	sufficient	/lib/security/pam_unix.so likeauth nullok
auth	required	/lib/security/pam_deny.so
account	required	/lib/security/pam_unix.so
password	required	/lib/security/pam_cracklib.so retry=3
password	sufficient	/lib/security/pam_unix.so nullok use_authok md5 shadow
password	required	/lib/security/pam_deny.so
session	required	/lib/security/pam_limits.so
session	required	/lib/security/pam_unix.so

Use system files

Prevent using weak passwords

Use md5 to store passwords in shadow



# Scenario 2 – LDAP auth with local backoff

- ▶ LDAP server provides directory with users
  - ◆ Identifiers, shell, email, name
  - ◆ Group membership
- ▶ saslauthd: provides interface with remote directory
  - ◆ User identifiers and attributes
  - ◆ Group membership

# Scenario 2 – LDAP auth with local backoff

```
ldap_servers: ldaps://dc1.DOMAIN.TLD
ldap_search_base: dc=DOMAIN,dc=TLD
ldap_bind_dn: cn=admin,ou=host,dc=DOMAIN,dc=TLD
ldap_bind_pw: Sup3rS3cr3TP4ssw0Rd
ldap_filter: (uid=%U)
ldap_scope: sub
#ldap_group_attr: memberUid
#ldap_group_match_method: filter
#ldap_group_filter: (memberUid=%u)
#ldap_group_search_base: ou=group,dc=DOMAIN,dc=TLD
#ldap_size_limit: 0
ldap_tls_check_peer: yes
ldap_tls_cacert_file: /etc/ldap/certs.txt
ldap_tls_cacert_dir: /etc/ssl/certs/
ldap_time_limit: 15
ldap_timeout: 15
ldap_version: 3
```

**Specificies where server can be found**

**bind\_dn is a system account to query the LDAP**

**Group membership mapping can be set**

# Scenario 2 – LDAP auth with local backoff

```
auth      required    /lib/security/pam_env.so
auth      sufficient  /lib/security/pam_unix.so likeauth nullok
auth      sufficient  /lib/security/pam_ldap.so use_first_pass
auth      required    /lib/security/pam_deny.so

account   required    /lib/security/pam_unix.so
account   [default=bad success=ok user_unknown=ignore] pam_ldap.so

password  required    /lib/security/pam_cracklib.so retry=3
password  sufficient  /lib/security/pam_unix.so nullok use_authtok md5 shadow
password  sufficient  /lib/security/pam_ldap.so
password  required    /lib/security/pam_deny.so

session   required    /lib/security/pam_limits.so
session   required    /lib/security/pam_unix.so
session   optional    /lib/security/pam_ldap.so
```

**Try local accounts (for offline admin) and then LDAP**

**Allow changing the password of both local and remote accounts**

# Scenario 3 – MS AD auth with local backoff

- ▷ MS AD server provides directory with users
  - ◆ Identifiers, shell, email, name
  - ◆ Group membership
- ▷ Machine must be enrolled into domain
  - ◆ Requires login using admin credentials
- ▷ sssd: System Security Services Daemon
  - ◆ Provides and caches information from remote AD system

# Scenario 3 – MS AD auth with local backoff

```
[sssd]
domains = DOMAIN.TLD
config_file_version = 2
services = nss, pam
default_domain_suffix = DOMAIN.TLD

[domain/DOMAIN.TLD]
default_shell = /bin/bash
krb5_store_password_if_offline = True
cache_credentials = True
krb5_realm = DOMAIN.TLD
realmd_tags = manages-system joined-with-adcli
id_provider = ad
fallback_homedir = /home/%u@%d
ad_domain = DOMAIN.TLD
use_fully_qualified_names = True
ldap_id_mapping = True
access_provider = simple
simple_allow_groups = group-admins
```

**Typical SSSD configuration at  
/etc/sss/sss.conf**

**Specified Domain and  
Kerberos5 configurations**

**Kerberos is the effective  
authentication protocol issuing  
authnz tickets**

**Supports MFA and HSM**

# Scenario 3 – MS AD auth with local backoff

auth	[success=2 default=ignore]	pam_unix.so nullok
auth	[success=1 default=ignore]	pam_sss.so use_first_pass
auth	requisite	pam_deny.so
auth	required	pam_permit.so
auth	optional	pam_cap.so

account	[success=1 default=ignore]	pam_unix.so
account	[default=bad success=ok user_unknown=ignore]	pam_sss.so
account	required	pam_permit.so

password	requisite	pam_pwquality.so retry=3
password	[success=2 default=ignore]	pam_unix.so obscure use_authtok try_first_pass yescrypt
password	sufficient	pam_sss.so use_authtok
password	requisite	pam_deny.so
password	required	pam_permit.so

session	required	pam_unix.so
session	optional	pam_sss.so
session	optional	pam_systemd.so
session	optional	pam_mkhomedir.so
session	required	pam_permit.so

**Try local accounts (for offline admin) and then AD**  
**Deny the rest**  
**Clear errors**  
**Set inheritable capabilities**

**Set Session Settings from local conf, AD**  
**Register session in systemd**  
**Create home directory**