Revocation Methods

Explicit:

- CRL - Certificate Revocation List
  - Sources: CRL-DP, indirect CRL, dynamic CRL-DP
  - Delta-CRL, windowed CRL, etc.
  - Certificate Revocation Tree (CRT) and other Authenticated Data Structures
- OCSP – On-line Certificate Status Protocol

Implicit:

- CRS - Certificate Revocation System

Open Questions

- Consistency between CRL and OCSP responses
  - It is possible to have a certificate with two different statuses.
- If OCSP is more timely and provides the same information as CRLs, do we still need CRLs?
- Which method should come first - OCSP or to CRL?

Revocation Methods

Explicit:

- CRL - Certificate Revocation List
  - Sources: CRL-DP, indirect CRL, dynamic CRL-DP
  - Delta-CRL, windowed CRL, etc.
  - Certificate Revocation Tree (CRT) and other Authenticated Data Structures
- OCSP – On-line Certificate Status Protocol

<u>Implicit:</u>

- CRS - Certificate Revocation System

Implicit Revocation: Certificate Revocation System (CRS)

- Proposed by Micali (1996)
- Aims to improve CRL communication costs
- Basic idea: CA periodically refreshes valid certificates
- Uses off-line/on-line signature scheme to reduce update cost

One-Way Hash Chains

- Versatile cryptographic primitive
- Construction:
    1. Pick random number $\mathbf{Y_N}$ and a public hash function **H()**
    2. Compute N values $\mathbf{Y_{N-1},\ldots,Y_0}$ such that $\mathbf{Y_{i-1} = H(Y_i)}$
    3. Secret $\mathbf{ROOT=Y_N}$, public $\mathbf{ANCHOR=Y_0}$

$$\mathbf{Y_0} \xleftarrow{H} Y_1 \xleftarrow{H} Y_2 \xleftarrow{H} \bullet\bullet\bullet \xleftarrow{H} Y_{N-1} \xleftarrow{H} \mathbf{Y_N}$$

- Properties:
    - Use in reverse order of construction: $\mathbf{Y_0}$, $\mathbf{Y_1}$, …, $\mathbf{Y_N}$
    - Hard to compute $\mathbf{Y_i}$ from $\mathbf{Y_j}$ (if j<i), easy to compute $\mathbf{Y_j}$ from $\mathbf{Y_i}$
    - For example: easy to compute $\mathbf{Y_1}$ from $\mathbf{Y_2}$ since $\mathbf{Y_1=H(Y_2)}$
    - But, Infeasible to compute $\mathbf{Y_2}$ from $\mathbf{Y_1}$
- Verifier can efficiently authenticate $\mathbf{Y_j}$ knowing $\mathbf{Y_i}$ (j<i) by verifying whether: $\mathbf{Y_j} = H^{i-j}(\mathbf{Y_i}) = H(H(\ldots H(\mathbf{Y_i})\ldots))$
- This method is robust to missing values

CRS: Creation of a Certificate

- Two new parameters included in each PKC: $\mathbf{Y_0}$ and $\mathbf{N_0}$

$$\mathbf{Y_0} = H^{MAX}(\mathbf{Y_{MAX}})$$

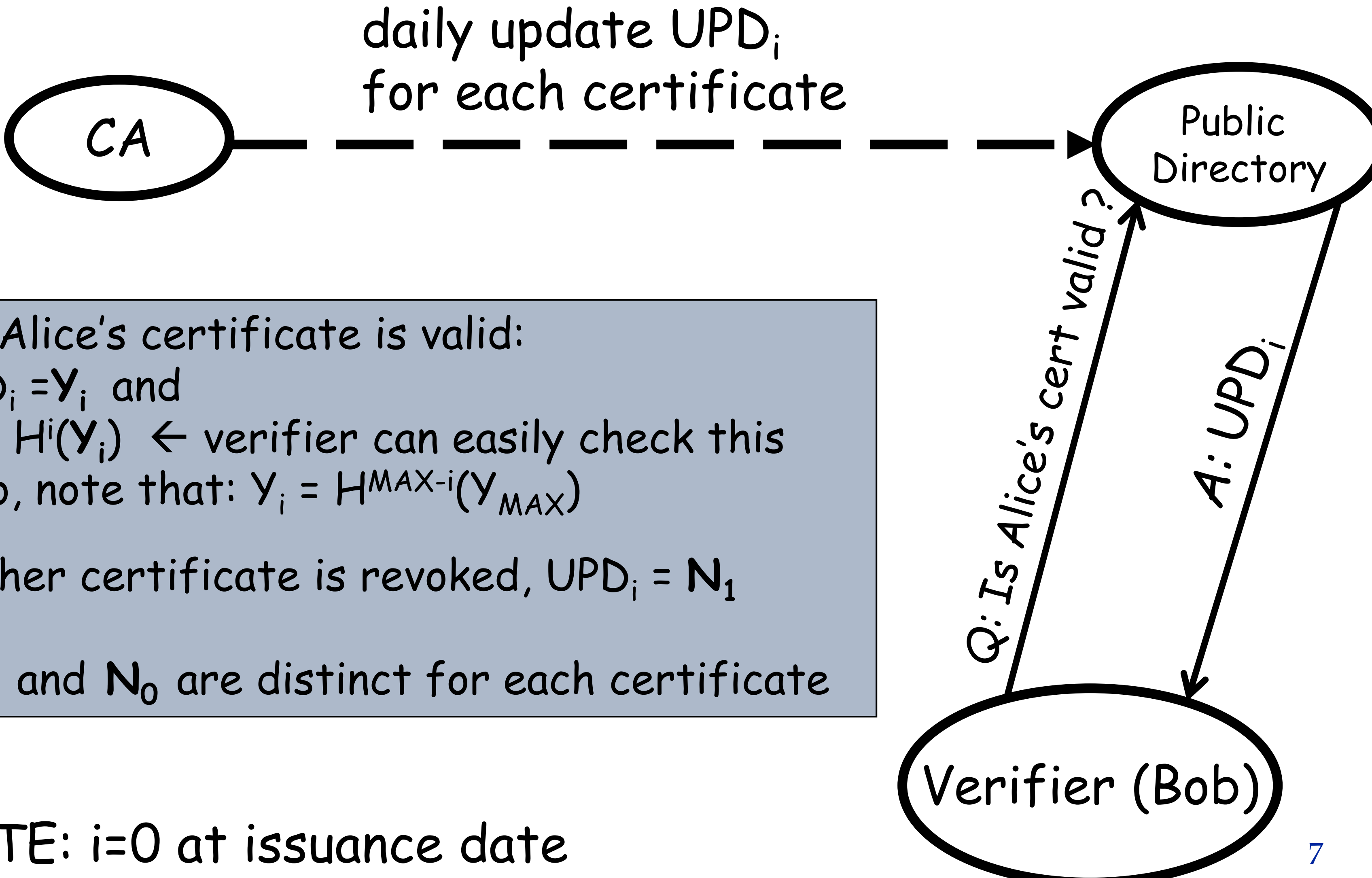
CHAIN ANCHOR

CHAIN ROOT

$$\mathbf{N_0} = H(\mathbf{N_1})$$

- [$\mathbf{Y_{MAX}}$,$\mathbf{N_1}$] -- per-PKC secrets stored by CA
- H() -- public one-way function, e.g., SHA-2

CRS Example:
Certificate issued for a year, refreshed daily



- If Alice's certificate is valid:
- $UPD_i = \mathbf{Y}_i$ and
- $\mathbf{Y}_0 = H^i(\mathbf{Y}_i)$ ← verifier can easily check this
- Also, note that: $Y_i = H^{MAX-i}(Y_{MAX})$

- If her certificate is revoked, $UPD_i = \mathbf{N}_1$

- $\mathbf{Y}_0$ and $\mathbf{N}_0$ are distinct for each certificate

NOTE: i=0 at issuance date

Lecture 13

Access Control

[lecture slides are adapted from previous slides by Prof. Gene Tsudik]

Recall: Security Services

- Confidentiality: to assure information privacy and secrecy
- Authentication: to assert who created or sent data
- Integrity: to show that data has not been altered
- Access Control: prevent misuse of resources = control access to them

  e.g., files, directories, accounts, printers, computers, IoT devices, etc.
- Availability: to offer access to resources, permanence, non-erasure

Access Control (AC)

- **A “language” for expressing access control policies: who can access what, how and when …**

- **Enforcement of access control**
  - Identify all resources (objects) and their granularity
  - Identify all potential users (subjects)
  - Specify rules for subject/object interaction
  - Guard them in real time

Model and Terminology

- **Subjects:** users or processes

- **Objects:** resources (files, memory, printers, routers, plotters, disks, processes, etc., etc.,...)

Focus of Access Control

- **What a subject is allowed to do**

- **What may be done with an object**

Access Modes

- **“Look” at an object, e.g.:**
  - Read file
  - Check printer queue
  - Read screen
  - Query database
  - Turn on/use microphone, etc., etc.

- **“Change” an object, e.g.:**
  - Write/append/erase file
  - Print on a printer
  - Display on screen
  - Use speakers (audio out)
  - Send packets via WiFi/Bluetooth, etc., etc.

Access Modes: Bell-Lapadula model

execute, read, append, and write

	Execute	Read	Append	Write
Observe		X		X
Alter			X	X

UNIX/Linux/*x Operating Systems

- **execute:** execute (program) file, search directory
- **read:** read from file, list directory
- **write:** write (re-write or append) file, create or rename file in directory

Example: Windows NT/2000 (NTFS)

- **execute**
- **read**
- **write**
- **delete**
- **change permission**
- **change ownership**

AC Types

Who is in charge of setting AC policy?

- **Discretionary:** resource owner
- **Mandatory:** system-wide policy

Access Control Structures

i. **Access Control Matrix**

ii. **Capabilities**

iii. **Access Control Lists**

Access Control Matrix

Object

Subject

	Bill.doc	Edit.exe	Fun.com
Alice	{0}	{execute}	{execute,read}
Bob	{read,write}	{execute}	{execute,read,write}

Access Control Lists 1/2

Keep access rights to an object with that object:

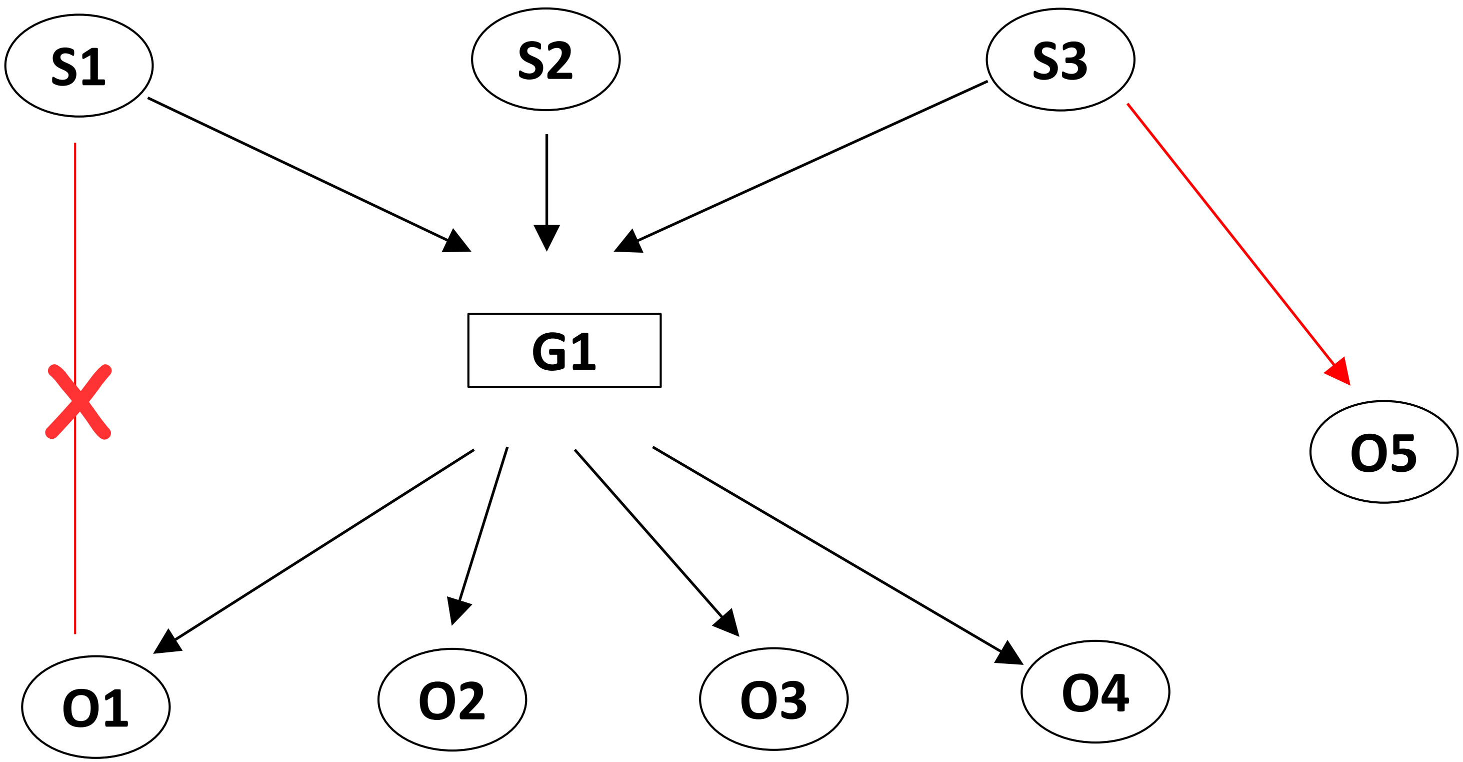
- **ACL for bill.doc:**
  - **Bob: read, write**
- **ACL for edit.exe:**
  - **Alice: execute;**
  - **Bob: execute**
- **ACL for fun.com:**
  - **Alice: execute, read;**
  - **Bill: execute, read, write**

- As many ACLs as there are objects
- Each ACL either signed or stored in protected place

Access Control Lists 2/2

- **Managing access rights can be difficult**
- **Groups can be helpful …**
- **Groups simplify definition of access control policies**

Access Control Lists



Capabilities 1/2

- **Capabilities are associated with discretionary access control**

- **Reason: difficult to get full view of who has permission to access an object**

- **Very difficult to revoke a capability – owners and objects have to keep track of all issued capabilities**

  - As many capabilities as there are (subject/object) pairs
  - Each capability either signed or otherwise protected
  - Hard to revoke in a distributed setting

Capabilities 2/2

Keep access rights with the subject:

- **Alice's capabilities:**
  - **[edit.exe:execute];**
  - **[fun.com:execute,read]**

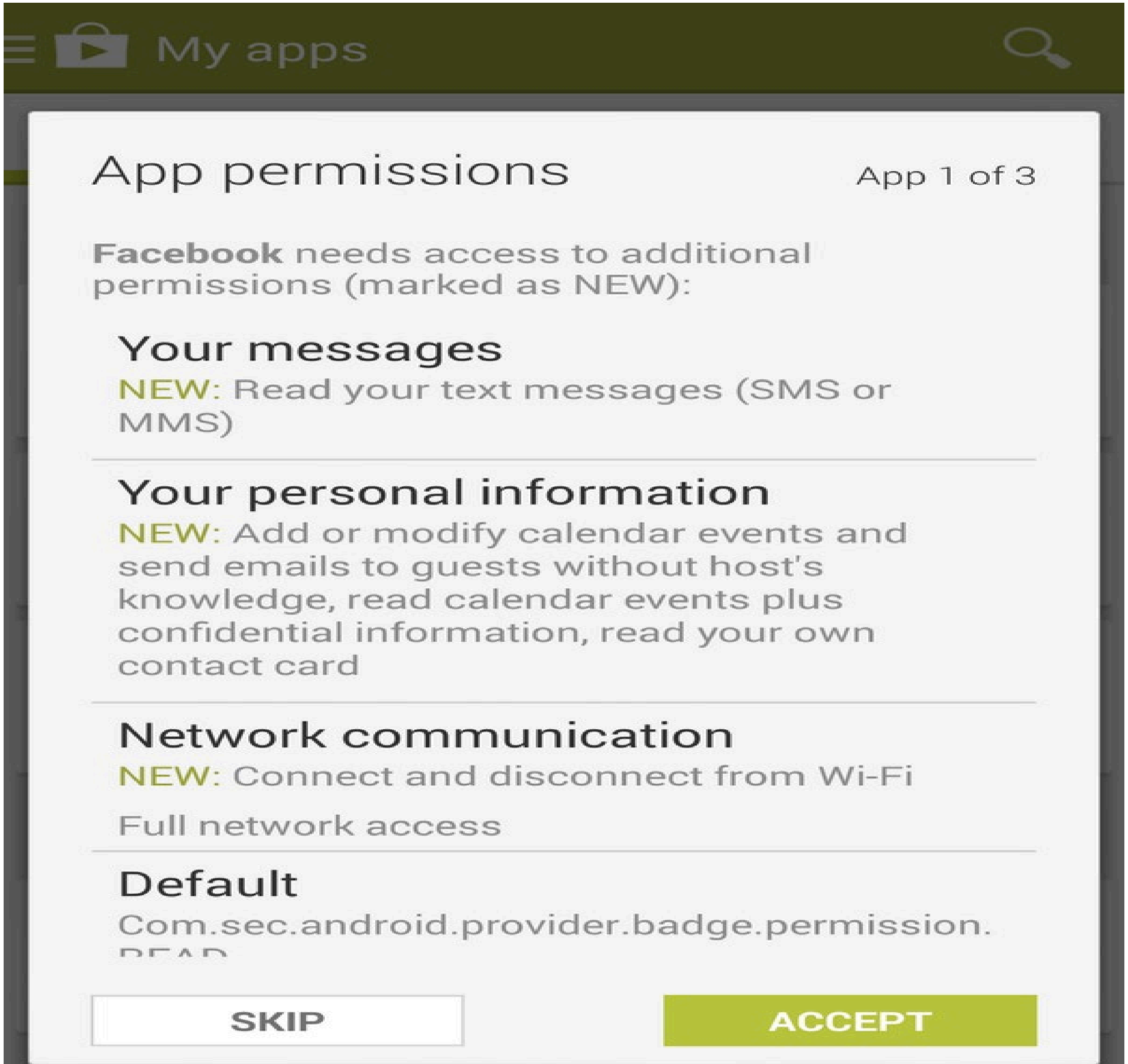
- **Bob's capabilities:**
  - **[bill.doc:read,write]**
  - **[edit.exe:execute]**
  - **[fun.com:execute,read,write]**

In Summary

- **Centralized Systems:**
  - **ACLs are better**

- **Distributed Systems:**
  - **Capabilities are better**

Example: Android Security/Permissions



Android Security Model

- Application-level permissions model
  - Controls access to app components
  - Controls access to system resources
  - Specified by the app writers and seen by the users

- Kernel-level sandboxing and isolation
  - Isolate apps from each other and the system
  - Prevent bypass of application-level controls
  - Relies on Linux Discretionary Access Control (DAC)
  - Normally invisible to the users and app writers

Discretionary Access Control (DAC)

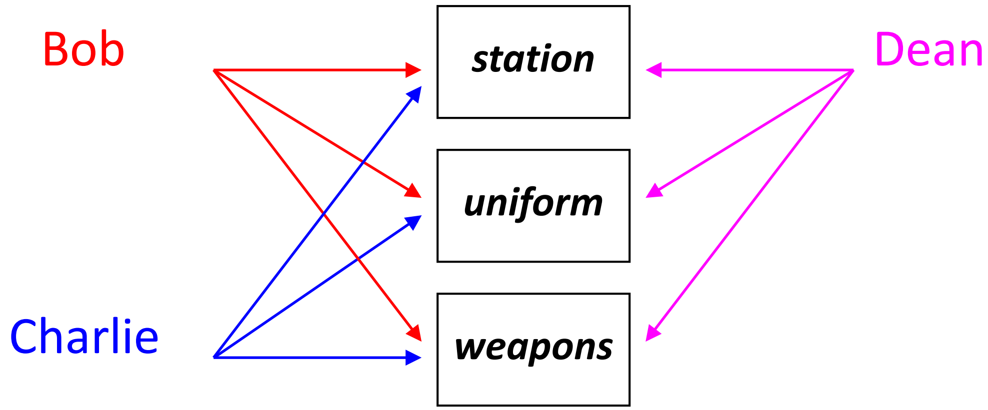
- Typical form of access control in Linux and many other Unix-derived OS-s
- Access to data is entirely at the discretion of the owner/creator of the data
- Some processes (e.g., uid 0) can override and some objects (e.g., sockets) are unchecked
- Based on user & group identity

ROLE BASED ACCESS CONTROL (RBAC)

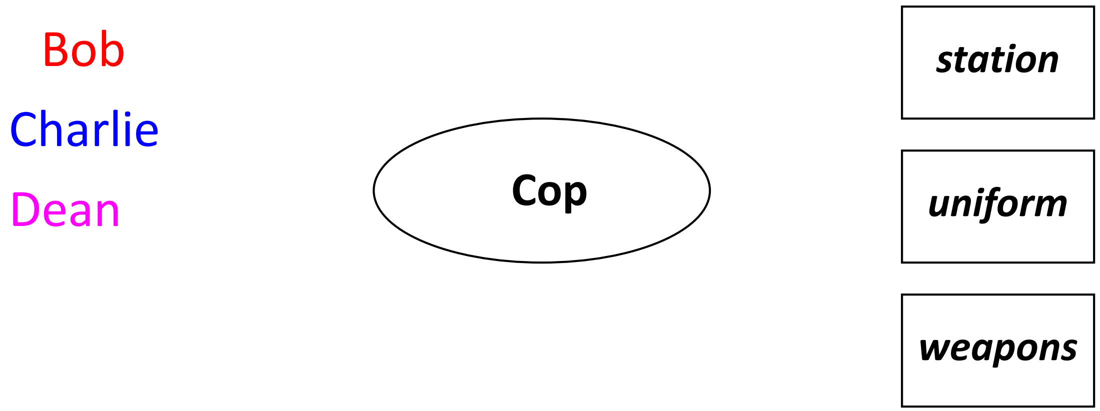
RBAC Basics

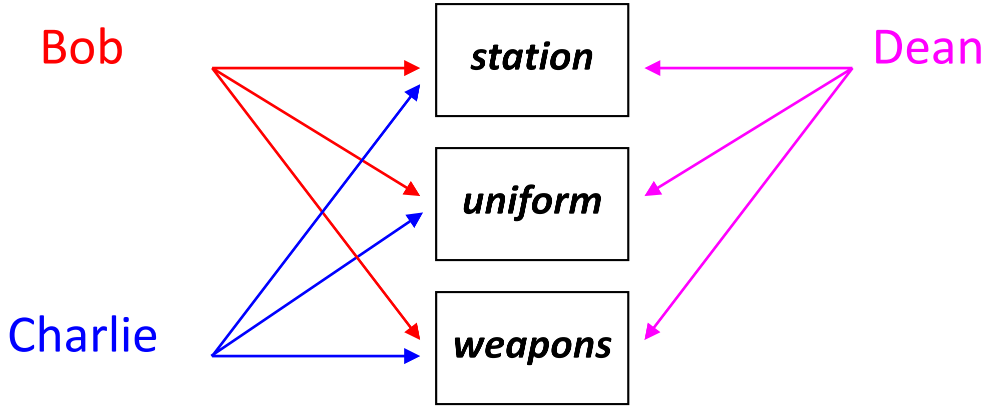
- Users are associated with roles
- Roles are associated with permissions
- A user has permission only if s/he has a role associated with that permission

Example: Cops (aka Police Officers) (User/Permission Association)

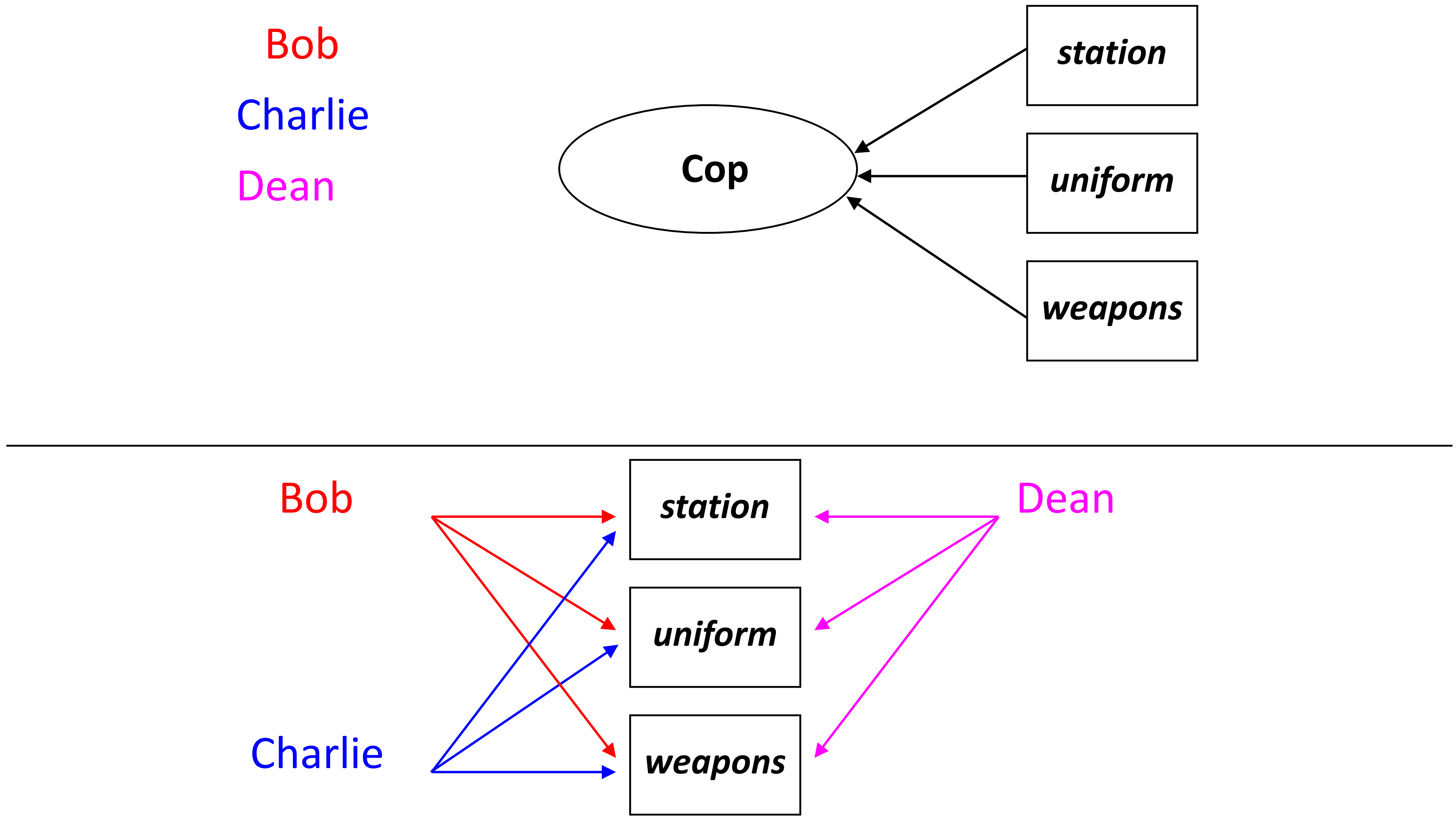


Example: RBAC

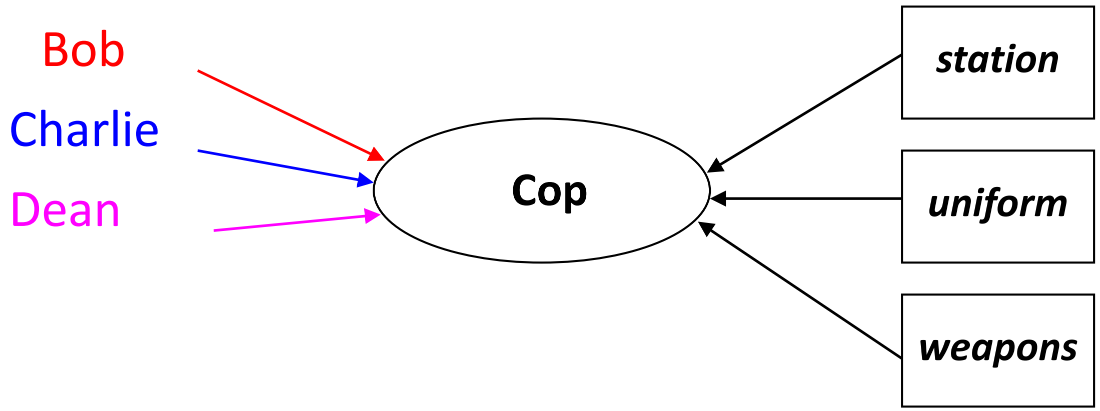




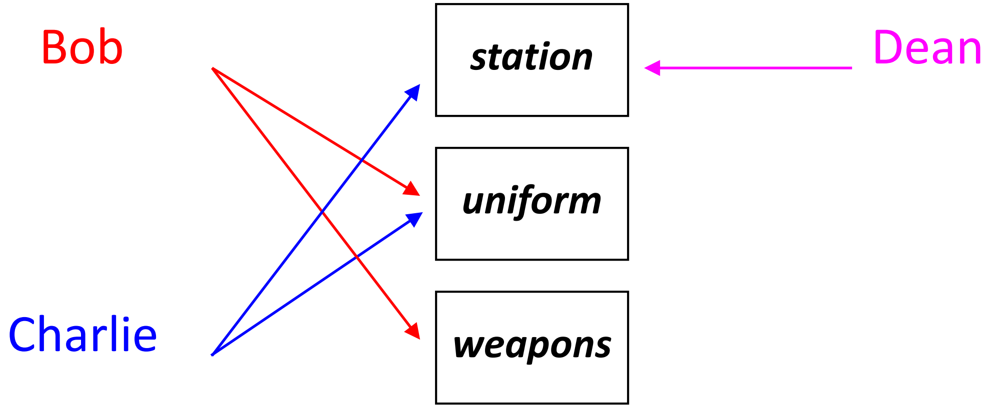
Example: RBAC



Example: RBAC



Here RBAC doesn't work …



Example: Alice becomes a Cop