

Macaroons and SciToken

Paul Millar

dCache Workshop 2018 at DESY, Hamburg; 2018-05-28
<https://indico.desy.de/indico/event/19920/>



Nordic e-Infrastructure
Collaboration



eXtreme DataCloud



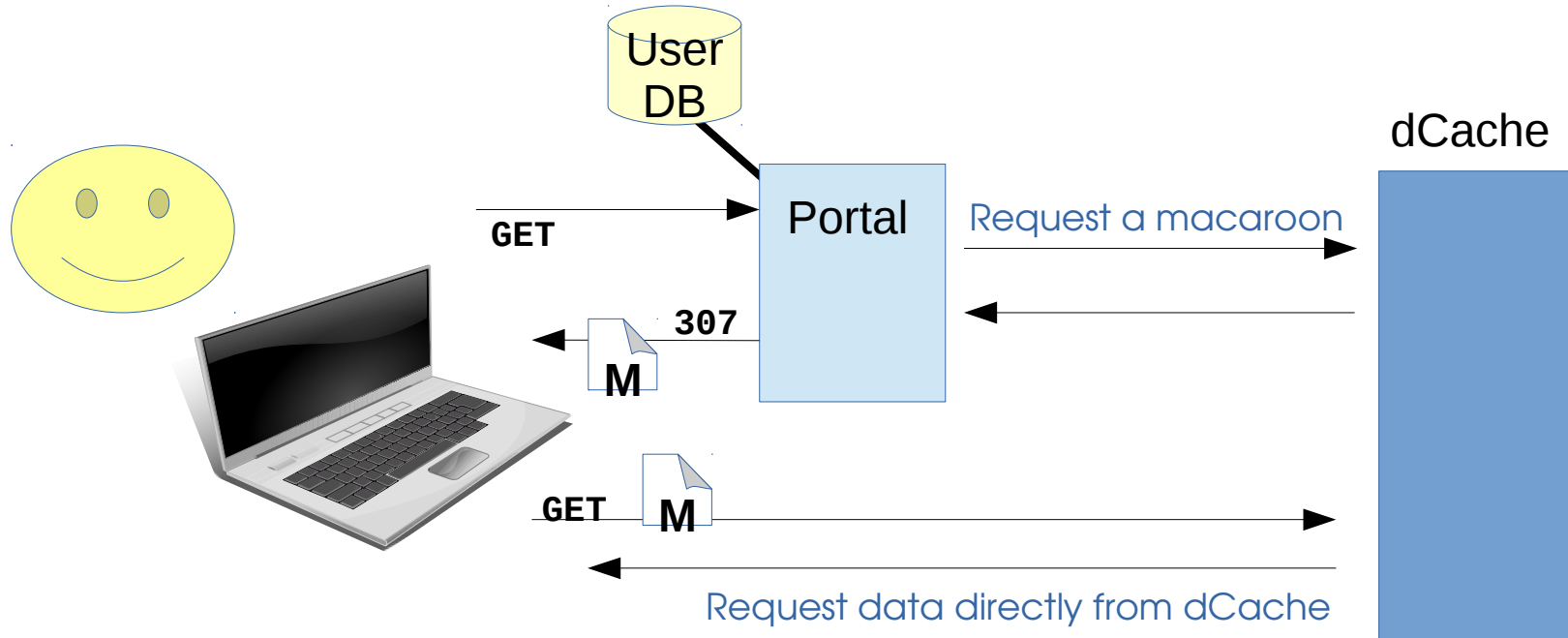
Introducing Macaroons



Macaroons “cheat-sheet”

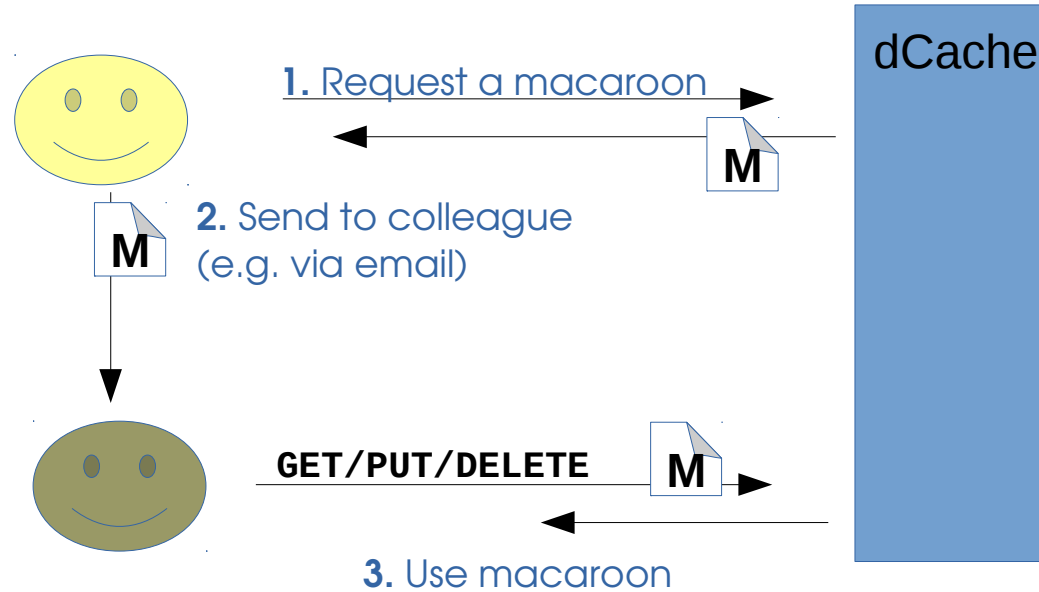
- Macaroon is a **bearer token**.
 - Macaroon contains zero or more **caveats**.
 - Each caveat **limits** something about the macaroon:
 - who** can use it,
 - when** they can use it, or
 - what** they do with it.
 - Anyone can **add a caveat** to a macaroon
 - ... creating a new, more limited macaroon.
 - No one can **remove a caveat** from a macaroon
-

What are macaroons good for?



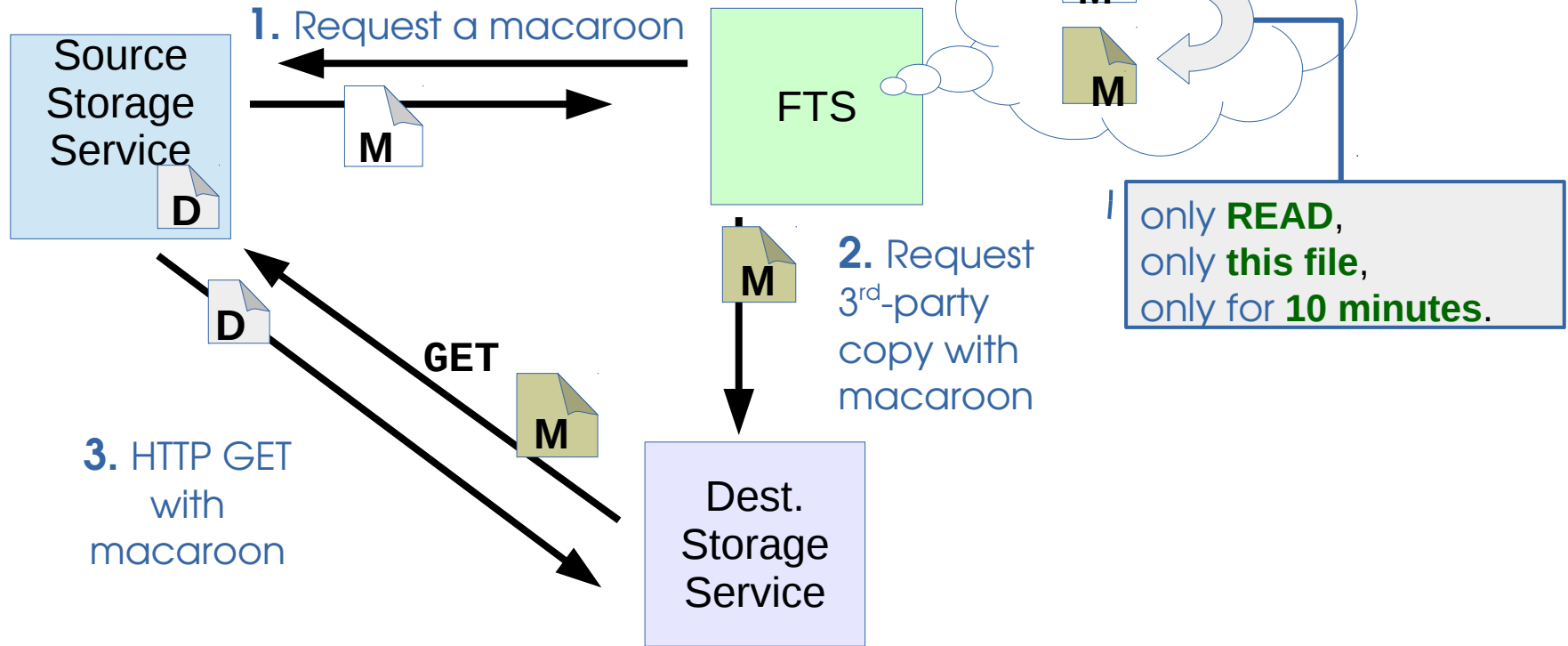
Community Portals

What are macaroons good for?



Delegating/Sharing

What macaroons good for?



Authorising third party copies

Future activity

- Add ability to cancel subset of macaroon
 - Add ability to request macaroon in REST API (frontend)
 - Add ability to use macaroon with more doors (dcap, ftp, ...)
-

SciTokens ... what they are

- They are OAuth2 tokens with defined scopes.
- The OAuth2 server is run by a VO
- The services trust the OAuth2 server
- The OAuth2 server issues a user with a token

The token does **not** identify the user.

Why both with SciTokens?

- Solve federated/distributed authorization
 - If the SciToken server says a user can write a file then ...
 - Authentication technology agnostic
 - The storage system does not care how the user authenticates – allows migration from X.509 to “something else”.
 - Adopting a commonly used, industry standard
 - Standing on the shoulders of giants ...
-

Potential problems with SciTokens

- Central point of failure
 - Needs to scale to support users
 - Generating SciTokens are relatively expensive
 - Services do not know who are their users
 - Traceability and user-banning difficult
 - Legal requirements difficult to enforce
 - No support for delegation
-

SciTokens vs macaroons: comparison cheat-sheet

- Who issues them
 - (SciToken: VO service, macaroons: dCache)
 - How expensive to generate?
 - (SciTokens a few Hz, macaroon: a few kHz)
 - Generate reduced token
 - (SciToken may be possible with request to OAuth2 service, macaroon possible without any communication)
-

Adding SciToken support in dCache

- Add OAuth support

Broadly similar to OpenID-Connect support that we already have.

- Add authz framework

Map all SciTokens from the same server to some specific “group” account; e.g., CMS SciToken → the “cms” user.

- Encode authz information in SciToken as a Restriction
-

dCache “under the hood”: Restrictions

- The door converts a valid macaroon into a logged-in user + a **Restriction**
 - Restrictions travel with PnfsManager requests
 - Restrictions **limit** what a user can do.
 - PnfsManager checks with the restriction to see if an operation is allowed.
 - Macaroon caveats correspond to Restrictions.
-

Next steps ...

- dCache part of Brian's FTP-less, X.509-less eco-system
CHEP demo.
 - Anticipate integrated support in dCache v5.0 or v5.1
-

Thanks for listening!

Backup slides

What macaroons good for?

There are lots more possibilities...

- Hiding authentication mechanism (e.g., X.509) from users
- Centralised authorisation service

...

Macaroons are a **basic building-block** that has many potential uses.

Enough theory, now for dCache ...

Getting a macaroon

- Unfortunately no standard way of doing this ... here's how with dCache
- Currently via the **HTTP/WebDAV door**:
- Request is **HTTP POST**:

- Must be SSL/TLS connection and include HTTP header:
Content-Type: application/macaroon-request

- Optional request body is JSON object, like:

```
{  
  "caveats": [ "caveat-1", "caveat-2", ...],  
  "validity": "<validity>"  
}
```

The **"caveats"** and **"validity"** fields are optional.
JSON object is optional → empty caveats and validity.

- If successful, response is JSON object with **macaroon** item

```
{  
  "macaroon": "MDAwZmxvY2F0a...."  
}
```

Using a macaroon

When authenticating **with dCache**:

- Standard HTTP request header:

Authorization: BEARER <macaroon>

- For awkward clients, embed macaroon in the URL:

https://webdav.example.org/mydir/file?authz=<macaroon>

For **3rd party HTTP transfers** (dCache authenticating with remote storage):

- **WebDAV** COPY request, add header:

TransferHeaderAuthorization: BEARER <macaroon>

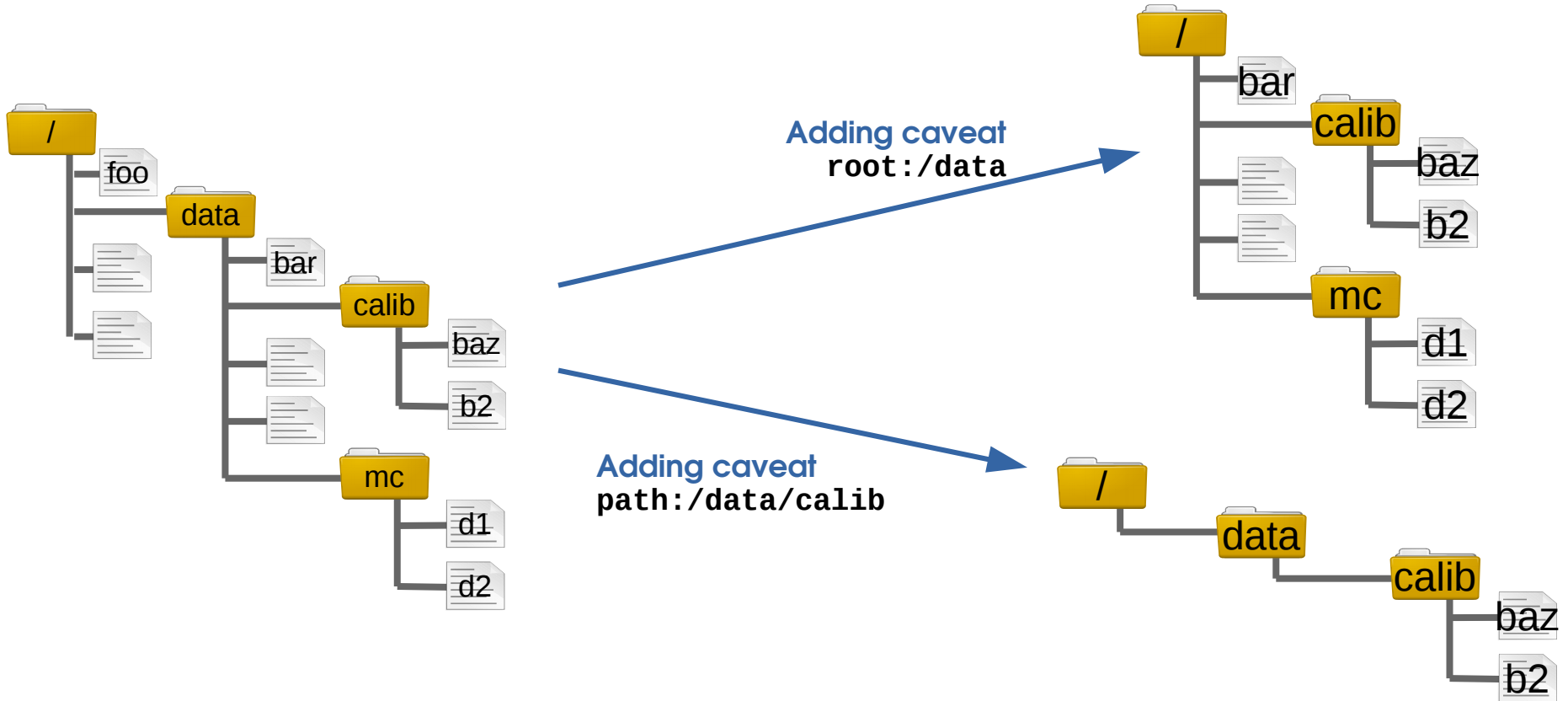
- **SRM** srmCopy request, include TExtraInfo; e.g.,

srmcp "-extraInfo=header-Authorization:BEARER <macaroon>"

Six caveats supported

- Unfortunately, there are no standard caveats. Here are those that dCache understands:
 - Three path caveats:
 - **root:** <path> – chroot into this directory,
 - **home:** <path> – the user's home directory (not currently used),
 - **path:** <path> – only show this path.
 - Two context caveats:
 - **before:** <timestamp> – when macaroon expires,
 - **ip:** <netmask list> – reduce which clients can use macaroon.
 - One permissions caveat:
 - **activity:** <comma-list> – what operations are allowed.
-

How path caveats affect namespace



Time caveat – expiring macaroon

before: <timestamp>

where <timestamp> is **ISO 8601 UTC time**;

e.g, **before:2017-05-29T16:00:00Z**

- Once time has elapsed, macaroon is useless.
- Validity can be reduced by adding more **before:** caveats.
- Short-cut: use the **validity** value in JSON request.

Calculates and adds a corresponding **before:** caveat.

The value is **ISO 8601 duration**; e.g., **PT3S** for 3 seconds.

Request JSON like **{"validity":"PT1M"}** returns a macaroon valid for 1 minute.

Client IP caveat – limit who can use it

ip:<netmask-list>

where **<netmask-list>** is a comma-separated list of subnets or addresses; e.g.,

```
ip:198.51.100.42,2001:db8:85a3::8a2:37:733, ↵  
192.0.2.0/24,2001:db8:cafe::/48
```

- Client's IP address must match (at least) one of the **ip: caveat's <netmask-list>**.
 - Adding more **ip: caveats** allows further restriction; e.g.,
ip:198.51.100.0/24 restrict to campus subnet
ip:198.51.100.28 only a specific machine
 - No **ip: caveats** means all clients may use the macaroon.
-

Activity caveats – limited what is allowed

activity:<activity-list>

where <activity-list> is a comma-separated list of allowed activities;

e.g.,

activity:DOWNLOAD,LIST

- Possible activities are:

DOWNLOAD, UPLOAD, DELETE, MANAGE, LIST, READ_METADATA, UPDATE_METADATA.

- Allowed activity may be further reduced by adding more **activity:** caveats.

No **activity:** caveat means client can do whatever the user requesting the macaroon can do.



Demo

- Show curl can upload and download files.
 - Show web-browser can see whole namespace.
 - Create an unrestricted macaroon.
 - Show curl upload/download and web-browser work with macaroon.
 - New macaroon with caveats:
activity:DOWNLOAD,LIST
path:/path/to/myfile
time < +5 minutes
 - Share modified macaroon with audience as QR code.
 - Browse in web-browser with macaroon; use curl to show download works, upload doesn't.
 - Wait for timeout.
 - Show macaroon doesn't work any more (ask audience to verify)
-

Combining caveats: namespace

- The **root** caveats and **path** caveats combine to create a more restricted caveat:

```
root:/foo  
root:/bar
```

equivalent to

```
root:/foo/bar
```

- The **path** and **home** caveats are relative to the effective root when declared:

```
home:/foo/bar/home  
root:/foo  
path:/bar/baz  
root:/bar
```

equivalent to

```
root:/foo/bar  
home:/home  
path:/baz
```

- Adding **root**: outside an existing **path**: results in a non-functioning macaroon.
 - Multiple **home**: caveats have last-one-wins.
-

Demo #2: root and path

- Show dCache has several directories with content.
 - Create read-only macaroon
 - Browser dCache name-space with this macaroon
 - New macaroon with **root:/path-1** caveat.
 - Browser dCache name-space
 - New macaroon with **path:/path-1/path-2/myfile** caveat
 - Show only **/path-1/path-2/myfile** is visible.
 - New macaroon from previous, with **root:/path-1**
-

Demo #3: expiry time

- Show dCache has several directories with content.
 - Create macaroon with **path:/path/to/file** caveat with expiry time five minutes in the future.
 - Create a count-down timer window for when macaroon expires
 - Show dCache can read the file OK.
 - Create a QR code and share it with the audience.
 - Ask audience to try to view the picture.
 - Continue talking until window times out.
 - When macaroon expires, show file cannot be read
-

Activity caveat – limit what can be done

- Format:

activity:<activities>

where **<activities>** is a comma-separated list: one or more of **LIST, DOWNLOAD, MANAGE, UPLOAD, DELETE, READ_METADATA, UPDATE_METADATA.**

- No caveat is the same as all activities:

activity:LIST, DOWNLOAD, MANAGE, UPLOAD, DELETE, READ_METADATA, UPDATE_METADATA

- Multiple caveats are allowed,

Subsequent caveats must be a subset of earlier caveats.

Root caveat – a bit like 'chroot'

- Format:
root:<path>
 - No root caveat is the same as **root:/**
 - User sees only files and directories under this path.
 - Multiple caveats are allowed,
Subsequent caveats are resolved relative to the previous caveat. Must not be inconsistent with any path caveat.
-

Home caveat – an initial directory

- Format:

home : <path>

- No home caveat is the same as **home : /**
- How this is used is protocol and client specific
- Multiple caveats are allowed,

Caveats are resolved relative to the current root. Value is automatically updated after a root caveat.

Path caveat – specific target

- Format:
path:<path>
 - No home caveat is the same as **path:/**
 - The paths of files and directories are unaffected, but only directories leading up to **<path>** or have **<path>** as a prefix are visible.
You don't want to change the URL, but only allow access to this URL.
 - Multiple caveats are allowed,
Caveats are resolved relative to the current path.
-

Quick recap



Authn



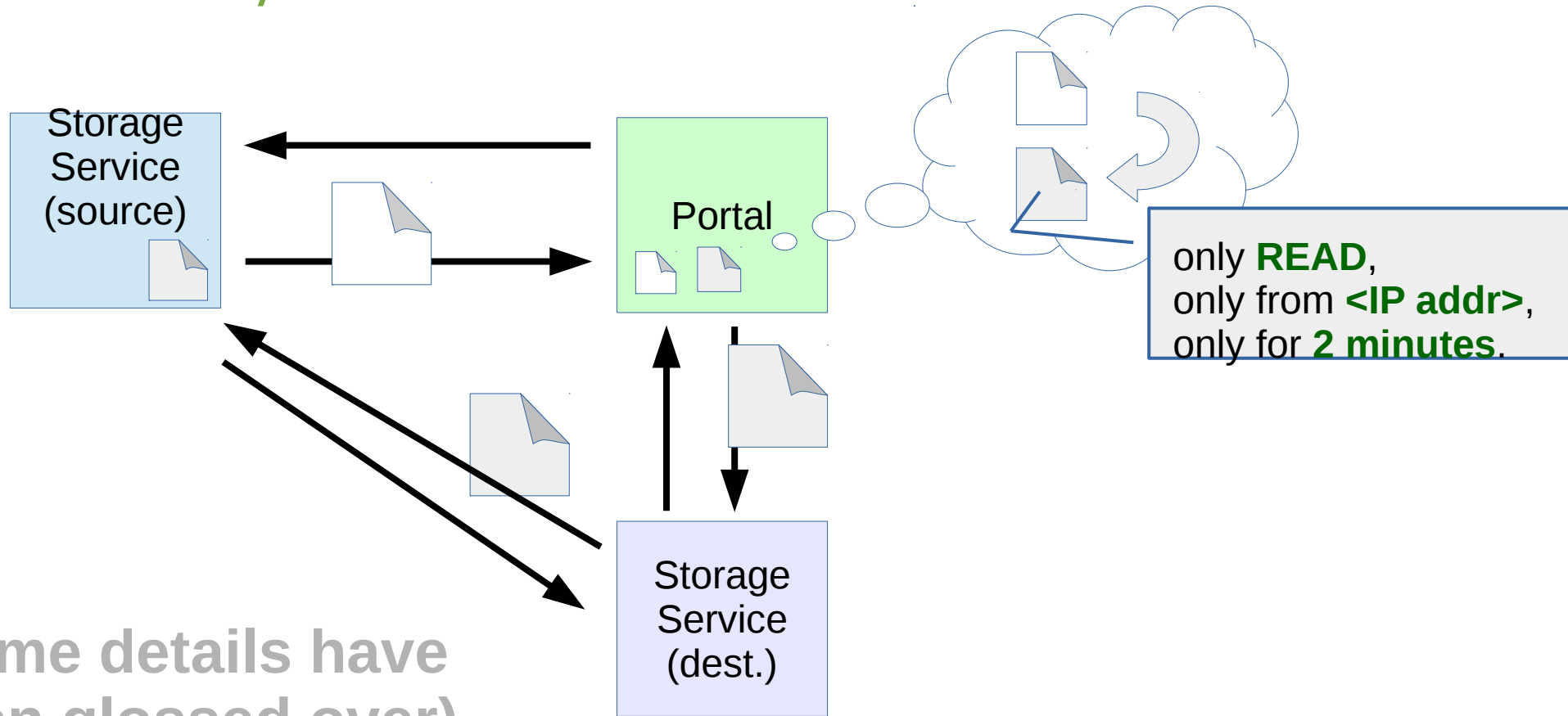
Authz

Authorisation without authentication?



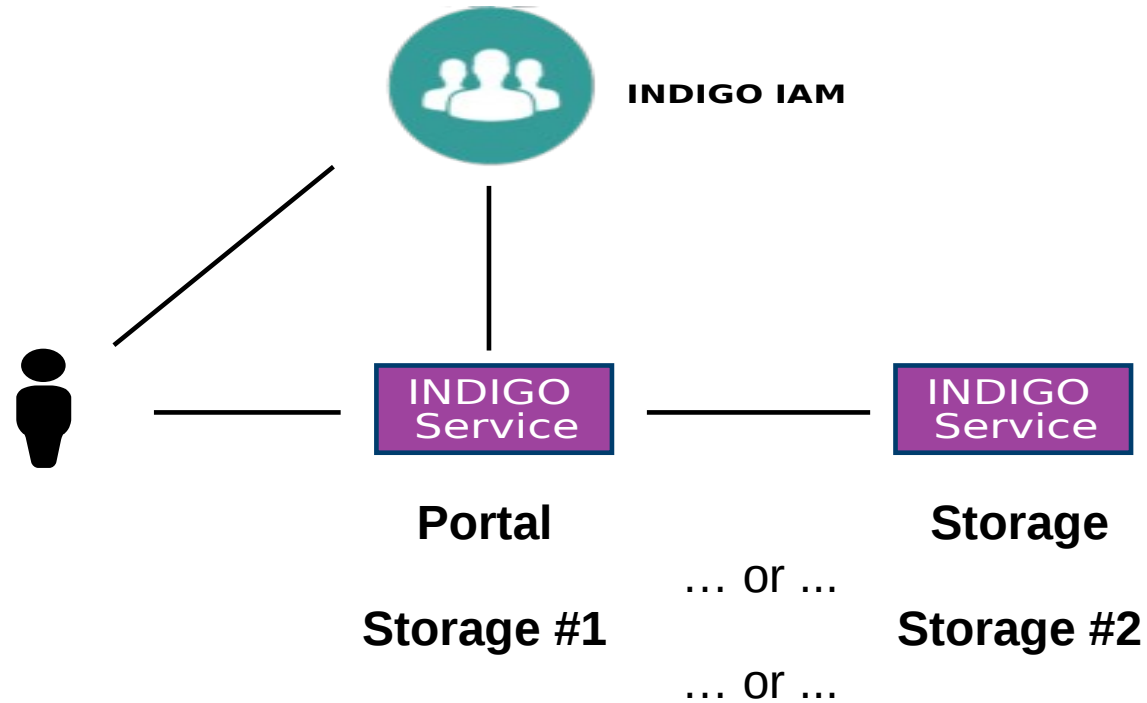
Photo by Alan Cleaver (CC-

Download / Share with macaroon



(some details have
been glossed over)

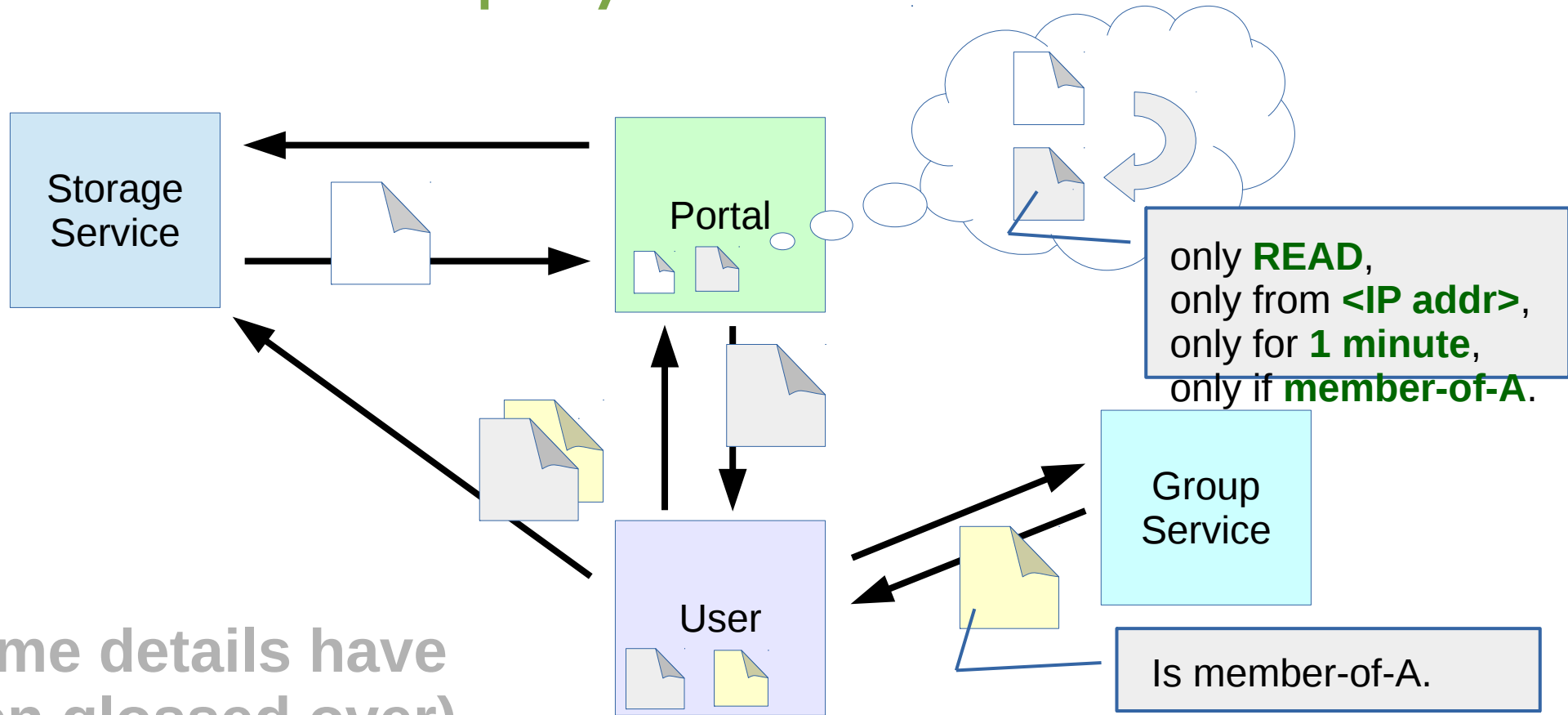
OpenID Connect delegation



3rd party caveats – extra cool!

- A 1st party caveat can be satisfied by the client.
 - A 3rd party caveat requires proof from some other service; e.g.
 - only **fred@facebook**,
 - only members of **VO ATLAS**,
 - only if not part of a **denial-of-service attack**.
 - The proof is another macaroon: a **discharge macaroon**.
-

Download with 3rd-party caveat



(some details have
been glossed over)

What are bearer tokens?

Bearer token is something the user presents with a request so the server will authorise it. There's no interaction between client and server.

Examples of bearer tokens:

HTTP BASIC authn, anything stored as a cookies.

Counter-examples:

- X.509 credential,
- SAML,
- Kerberos.



Group membership, too

- An OIDC provider can assert the user is a member of various groups
 - Group membership may require higher level of LoA:
 - For example, if the group is “loose collaboration” a site might require higher LoA; if the group is “commercial entity” a site might require lower LoA
-

One solution: a bearer token

