DARCs and Calypso

Non-fintech things to do with a blockchain

Intro

1) 3 Chapters
    a) DARC, Calypso, Auditing
2) For every chapter
    a) **Goals** - what you should be able to know and to do afterwards
    b) **Introduction** - theory and demonstrations
    c) **Exercices** - to be done in your group - later exercises build upon earlier ones!
    d) **Bonus** - if you're fast and waiting for the others - reading material and additional exercises

Ia. DARCs

- Understand how DARCs work
  - Rules: Action / Expression
  - Delegation - trust other DARCs
- Signature verification with DARCs
  - Need latest version
  - Parse the tree
- Recovery using DARCs

Ib. Delegation of Access

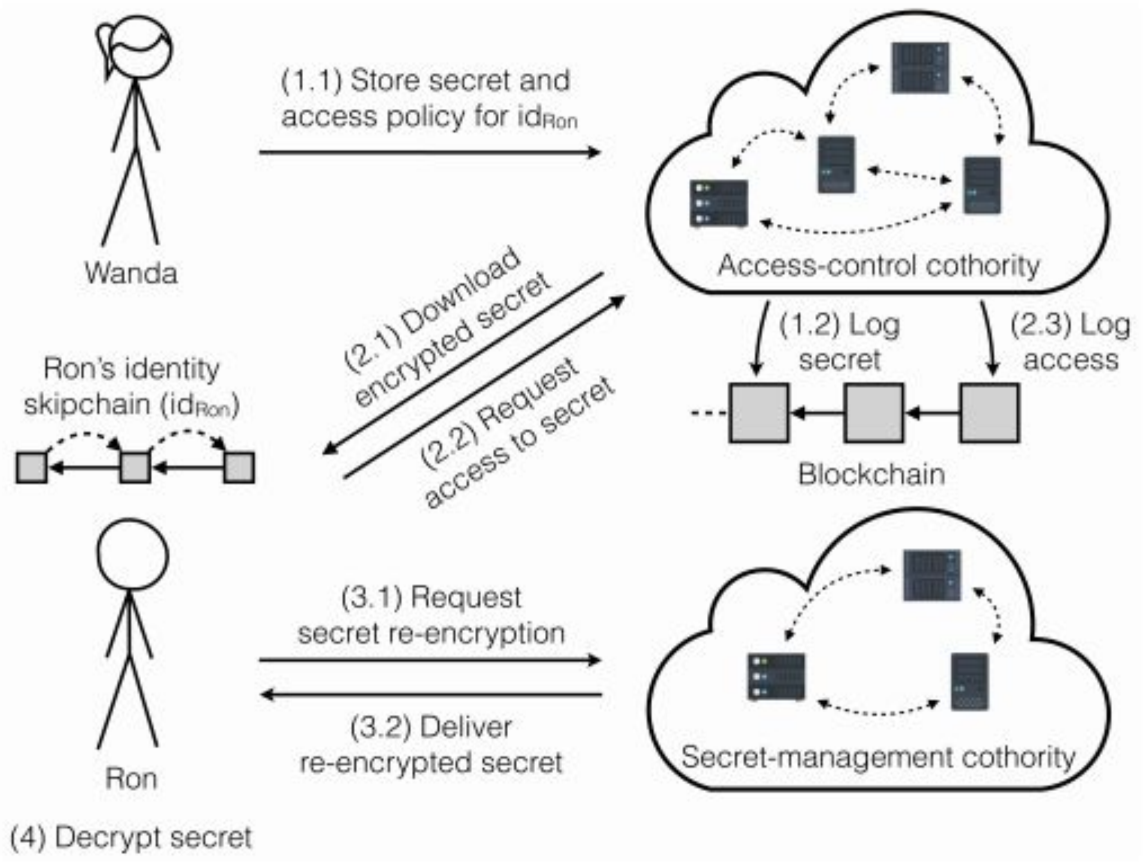
Replace the password with a *Delegation*

Full audit of updates and usage

Access can be updated over time

Ib. Calypso

2018/209 - Verifiable Management of Private Data under Byzantine Failures


Wanda
(1.1) Store secret and access policy for idRon
Access-control cothority
(1.2) Log secret
(2.3) Log access
Blockchain
Ron's identity skipchain (idRon)
(2.1) Download encrypted secret
(2.2) Request access to secret
Ron
(3.1) Request secret re-encryption
(3.2) Deliver re-encrypted secret
Secret-management cothority
(4) Decrypt secret

Ib. Authentication, Authorization, and Identity

Centralized

- Authentication
  - OpenID Connect
- Authorization
  - OAuth2
  - Links Authentication to Rights
- ActiveDirectory combines both
- Little control by the user
- Insider abuse very easy
- Closed-in systems - need to verify both servers and clients

https://www.town-crier.org/

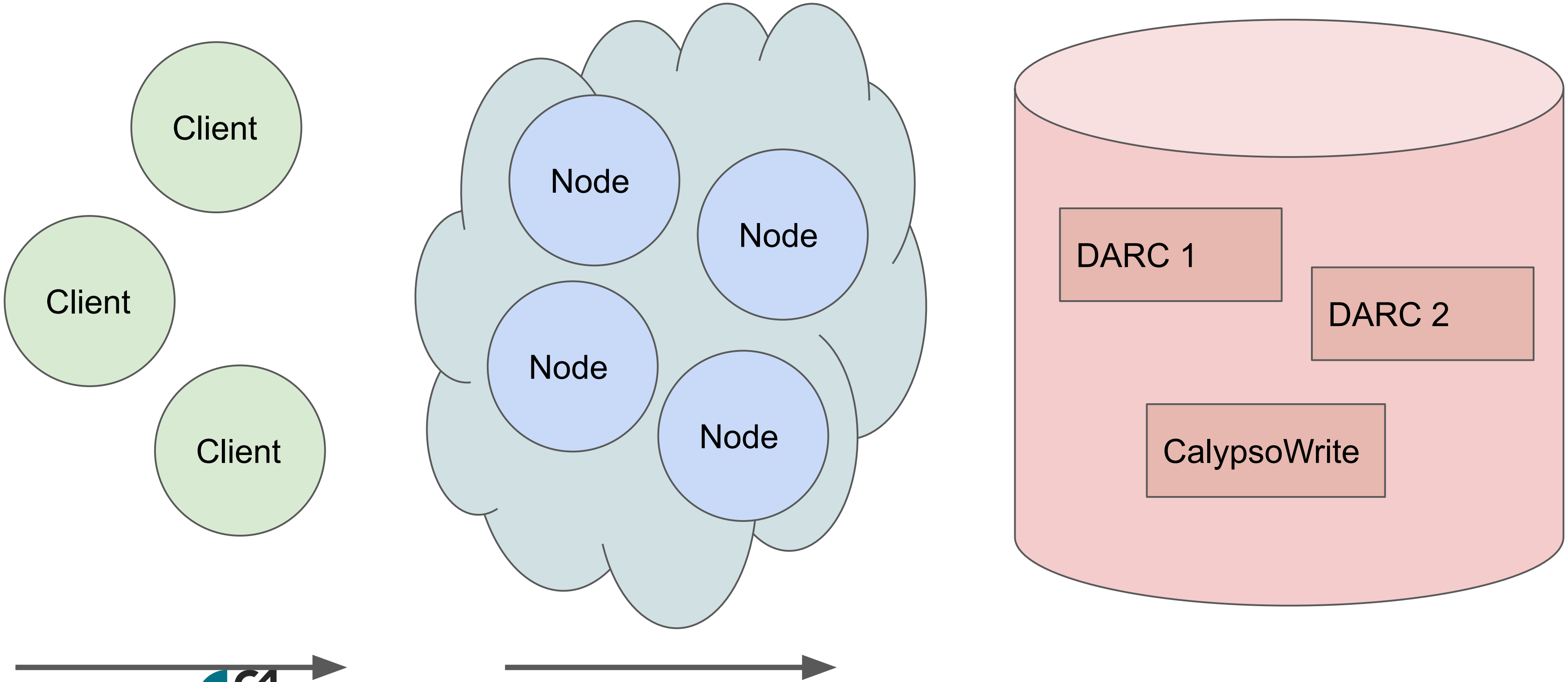
Decentralized / Distributed

- Authentication
  - W3C Decentralized IDs
- DARC -> Distributed Access Rights Control
- Is an attribute-based access control
- Current attributes supported
  - Signature authentication
    - Ed25519 / DID
  - AND / OR / () operations
  - Calls to smart contracts
- Self-sovereign identity
  - Add / remove devices
  - Key-rotation

Ib. DARC Rules

- A DARC combines the authorization, authentication, and delegation
- Every rule has
  - Action - the authorization described
  - Expression - how the action can be verified (authentication)
- Two special actions
  - **Evolve** - defines how the DARC can be changed from one version to the next
  - **Sign** - if this DARC has been delegated
- Available Expressions
  - AND / OR / ()
  - Call to view-method of a smart EVM contract
  - Signature verification (ed25519, W3C DID as WIP)
  - Delegation to another DARC

Ib. Digression - OmniLedger 101



Client

Client

Client

Node

Node

Node

Node

DARC 1

DARC 2

CalypsoWrite

Send Transactions
Create Queries

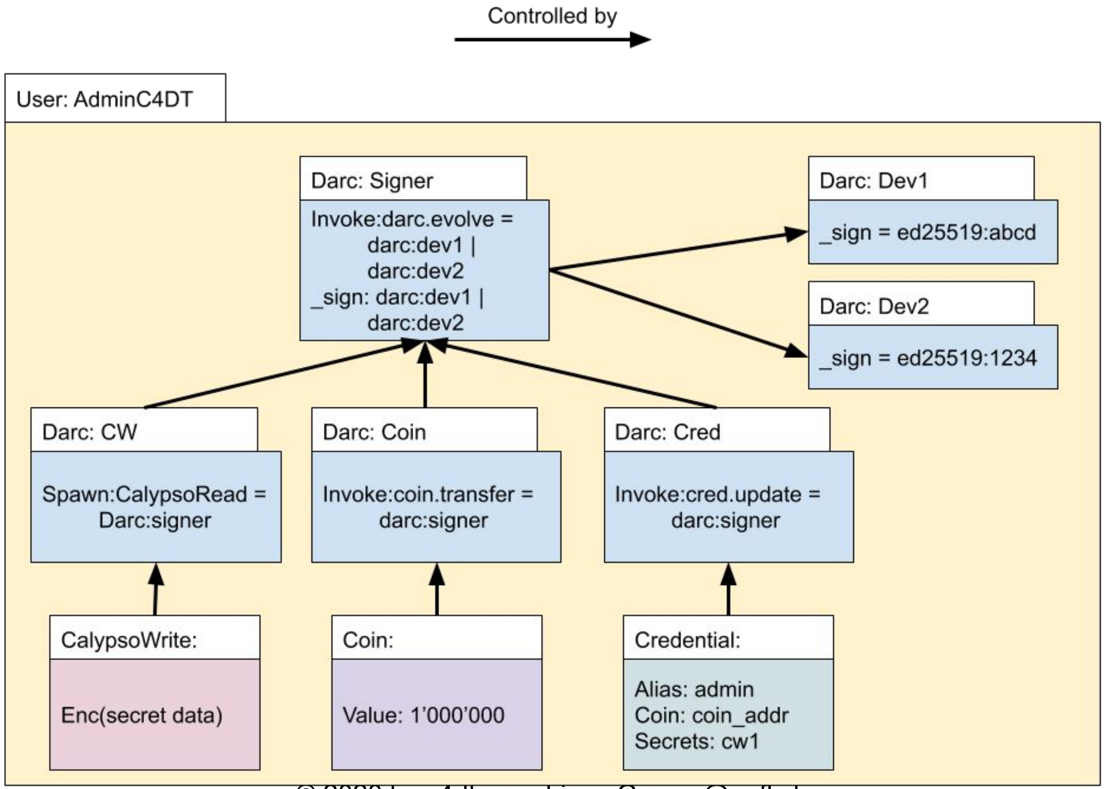
Create Consensus
Update state

Global State holds
Instances

Ib. Digression - OmniLedger 101

- ByzCoin - blockchain based on [1602.06997] - plus *time.Sleep(166)*
- OmniLedger - sharding (not implemented) - cool name
- Every Instance
  - Has a unique ID - 32 byte hash
  - Is linked to a contract (DARC, Coin, Value, CalypsoWrite, EVM, ...)
  - Links to a DARC for access control
  - Can be updated, depending on the contract
- Blocks contain transactions and a hash of the merkle tree of the instances
- Using forward-links, OmniLedger
  - Avoids forks
  - Allows for succinct proofs
- For more information, see GitHub - Stats - Article

Ib. Delegation Example - Inside User Definition



Controlled by

User: AdminC4DT

Darc: Signer

Invoke:darc.evolve =
darc:dev1 |
darc:dev2
_sign: darc:dev1 |
darc:dev2

Darc: Dev1

_sign = ed25519:abcd

Darc: Dev2

_sign = ed25519:1234

Darc: CW

Spawn:CalypsoRead =
Darc:signer

Darc: Coin

Invoke:coin.transfer =
darc:signer

Darc: Cred

Invoke:cred.update =
darc:signer

CalypsoWrite:

Enc(secret data)

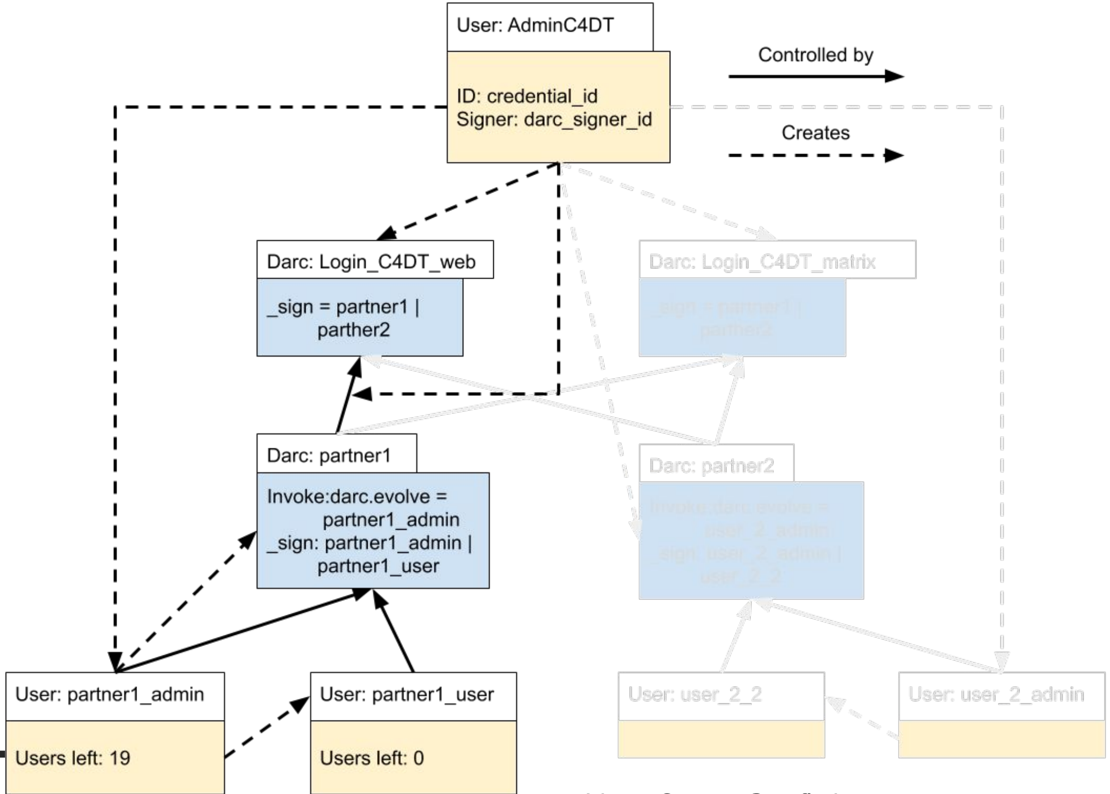
Coin:

Value: 1’000’000

Credential:

Alias: admin
Coin: coin_addr
Secrets: cw1

Ib. Federated Access Delegation



User: AdminC4DT

ID: credential_id
Signer: darc_signer_id

Controlled by

Creates

Darc: Login_C4DT_web

_sign = partner1 |
parther2

Darc: Login_C4DT_matrix

_sign = partner1 |
partner2

Darc: partner1

Invoke:darc.evolve =
partner1_admin
_sign: partner1_admin |
partner1_user

Darc: partner2

Invoke:darc.evolve =
user_2_admin
_sign: user_2_admin |
user_2_2

User: partner1_admin

Users left: 19

User: partner1_user

Users left: 0

User: user_2_2

User: user_2_admin

Ic. Exercise 1: Create a group DARC

- Each group creates a DARC with delegation to all members of the group
  - Decide which name the DARC should have
  - One member creates a new DARC (*Contacts / Create Group*)
    - adds all member of the group to the DARC using *Edit Group*
    - and sends the BaseID of the DARC to matrix
  - All the other members of the group add a link (*Contacts / Link Group*)
- Other members: try to modify the DARC
  - What is the error message?
  - Why does it fail?
  - How to change the DARC so that all members can modify the DARC?

Ic. Exercise 2: Add Recovery

- Allow your group-darc to recover your wallet
    - *Devices / Add Recovery* - chose the group DARC from exercise 1
- Try to recover an account from a member of your group
    - *Contacts / Recover*
    - You can open the link in a ‘private session’ in Chrome, but not in Firefox (missing DB-support)
- Try to recover an account from a member of another group - what happens?
- We’ll look in exercise 4 what happened to his/her account!
- Keep the recovery-account, so the members of your team can give you back your account, should you delete the private key!

Ic. Exercise 3: Start up StackBlitz

- For the next chapter, you’ll need more access to your account
- Create a new device, but don’t click on the link!: *Device / Add Device*
- Copy the URL somewhere safe
- Open StackBlitz
- On the left side, click ‘Settings’ (the gear-symbol), then change ‘Hot reload trigger’ to ‘Save’
- Paste the device-URL in the open file in the `deviceURL` variable
- Save (CMD+S or CTRL+S) to reload
- Open the file *handson/handson.component.ts*

Ic. Exercise 3b - Show Around in StackBlitz

- Left: files - only need *handson.component.ts*
- Middle: editor - usually stay in *ngOnInit*
  - Extra: for one time actions, add a *buttonYourCode* method and copy/paste code in *handson.component.html*
- Right: output of *ngOnInit* and the buttons
  - Blockchain explorer - Action buttons - Output
- Buttons:
  - *Print instance* - prints out the information about an instance
  - *Create CalypsoWrite* / *CalypsoRead* - used for later exercises
  - *UpdateLTS* - debugging - hopefully not used ;)
  - *Send Coin* - simple example of how a transaction is created
  - *Create Value / Update Value* - used for bonus exercise

Ic. Exercise 4: What Happened while Recovering?

- Use *Print instance* to show your ‘Signer DARC’
  - Count the number of delegations
  - Compare with the number of devices and recoveries
- Use the *Print instance* to show your ‘recovered by’ device
  - In the wallet, in the *Devices* tab, click on the name of the device to get the *baseID*

Id. Bonus

- Do the Id. Bonus
- Use the Stackblitz environment to create a new value instance, protected by your group darc
  - Ask another member of your group to update the value - for this you need to copy the *Send coin* button and modify it to

IIa. Calypso

- Explain the difficulty with classical encryption when things change
- Understand what re-encryption is
- Being able to store a secret and retrieve it

IIb. ‘Password’ Manager

Store key encrypted on blockchain

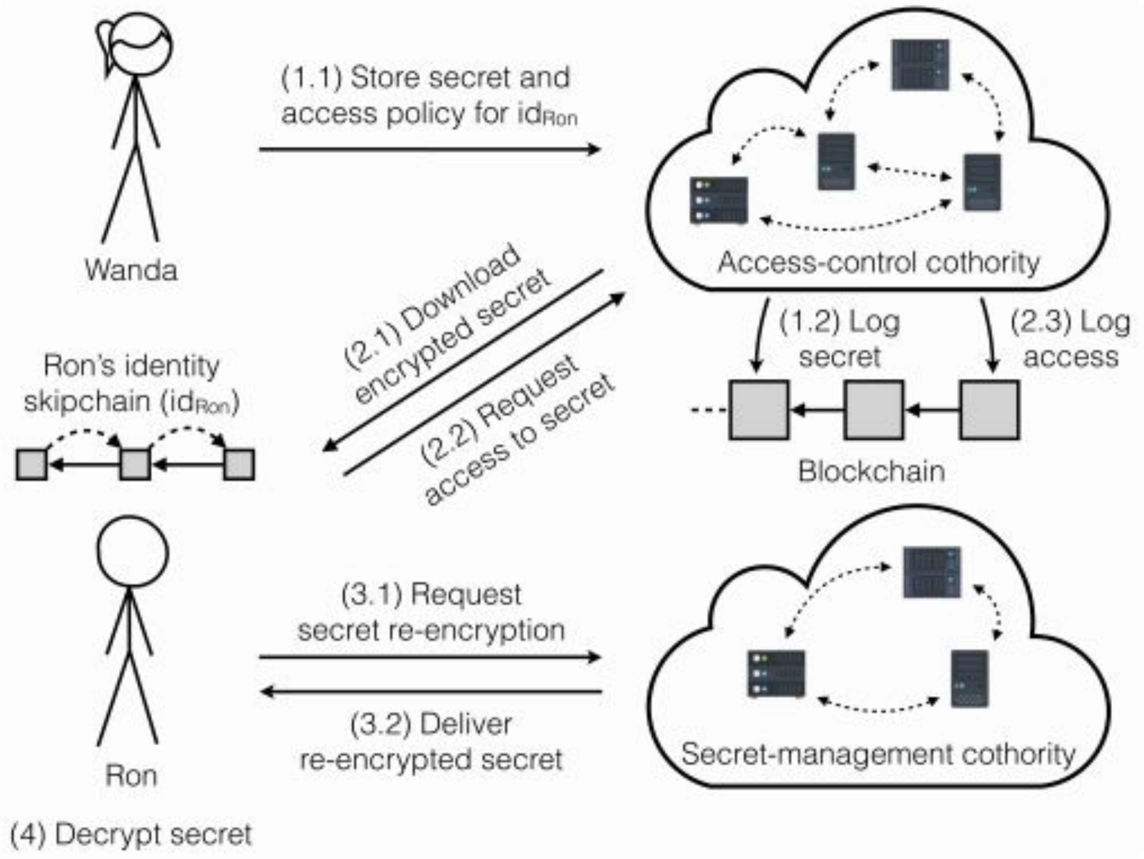
Only the block-chain can decrypt the key

Delegate access to the key

Decryption can be audited

IIb. Calypso

[2018/209 - Verifiable Management of Private Data under Byzantine Failures]


Wanda
(1.1) Store secret and access policy for id_{Ron}
Access-control cothority
(1.2) Log secret
(2.3) Log access
Blockchain
(2.1) Download encrypted secret
(2.2) Request access to secret
Ron's identity skipchain (id_{Ron})
Ron
(3.1) Request secret re-encryption
(3.2) Deliver re-encrypted secret
Secret-management cothority
(4) Decrypt secret

IIb.

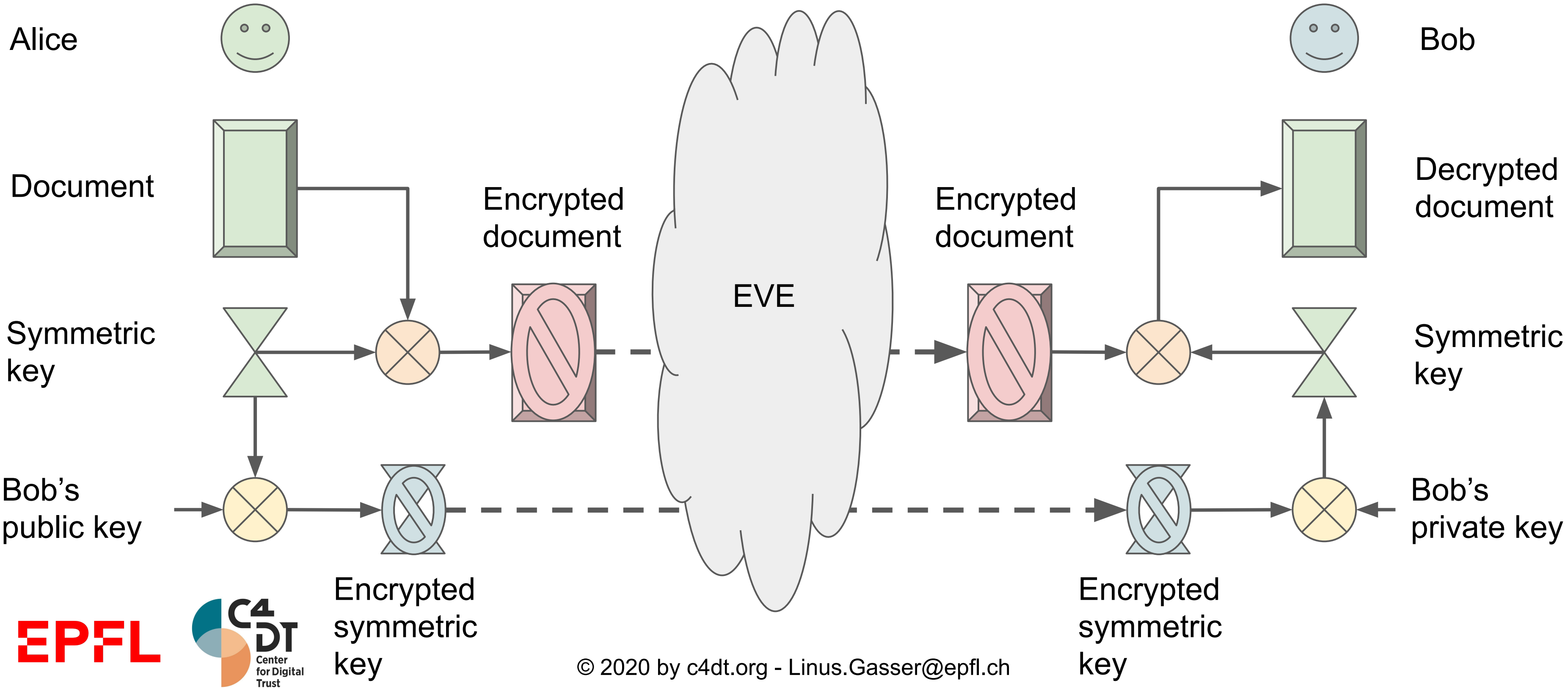
Asymmetric Encryption

- Public / Private keys
  - Private key needs to be kept safe
  - Public key is shared
- Challenge of Key Infrastructure
  - Public keys can be exchanged over unsafe medium
  - How to be sure that the keys are correct?
- Encryption
  - Everybody encrypts using the public key
  - Only the private key can decrypt
- Signature
  - Only the private key can sign
  - Everybody verifies using the public key
- Biggest downside: very slow!

Symmetric Encryption

- Only one type of key
- Needs to be treated securely
- Cannot be transferred as-is over unsafe medium
- Biggest upside: very fast!

IIb. Asymmetric and Symmetric Encryption



Alice

Document

Symmetric key

Bob's public key

Encrypted document

Encrypted symmetric key

EVE

Encrypted document

Encrypted symmetric key

Bob

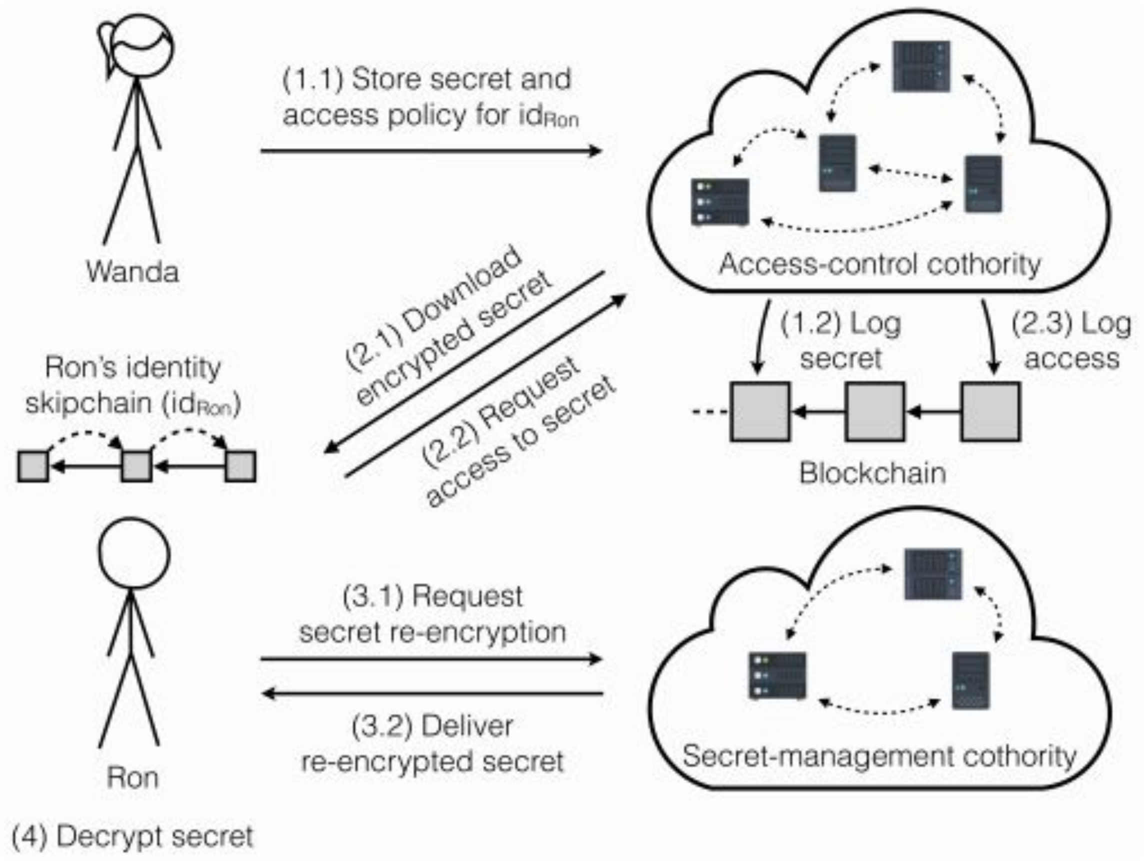
Decrypted document

Symmetric key

Bob's private key

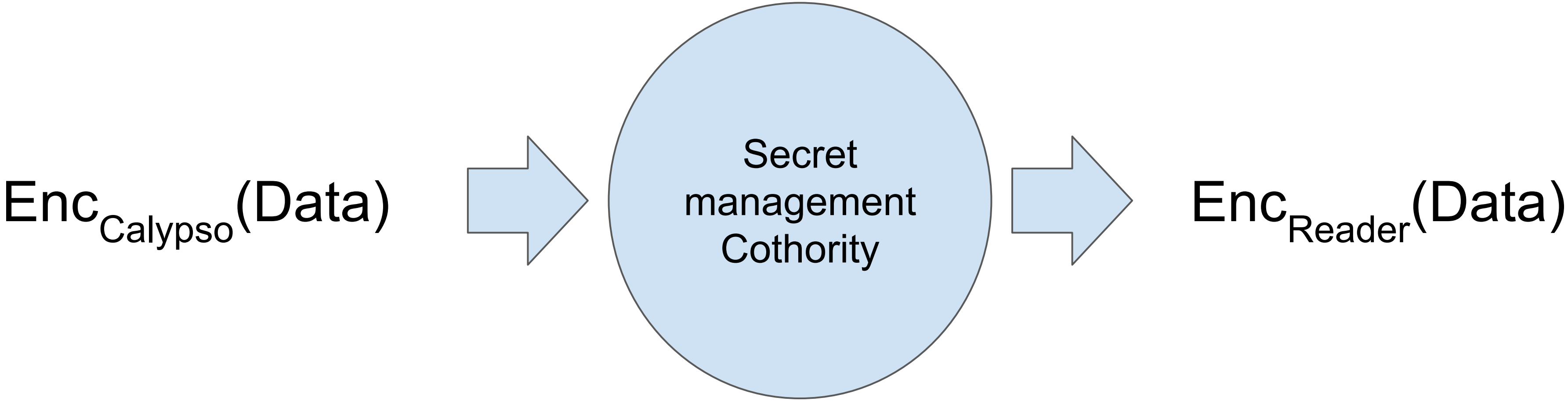
IIb. Calypso

2018/209 - Verifiable Management of Private Data under Byzantine Failures



Wanda
(1.1) Store secret and access policy for id$_{Ron}$
Access-control cothority
(1.2) Log secret
(2.3) Log access
Blockchain
(2.1) Download encrypted secret
(2.2) Request access to secret
Ron's identity skipchain (id$_{Ron}$)
Ron
(3.1) Request secret re-encryption
(3.2) Deliver re-encrypted secret
Secret-management cothority
(4) Decrypt secret

IIb. Calypso for Dummies



$\text{Enc}_{\text{Calypso}}(\text{Data})$

Secret management Cothority

$\text{Enc}_{\text{Reader}}(\text{Data})$

- How to
  - re-encrypt without decrypting the data?
  - proof the reader is allowed to re-encrypt?
  - avoid fraudulent re-encryption requests?
  - change access rights after encryption?

IIb. Re-encrypt Without Decrypting the Data?

$\mathrm{Enc}_{\mathrm{Calypso}}(\mathrm{Data})$

$\mathrm{Enc}_{\mathrm{P1+P2+P3}}(\mathrm{Data})$

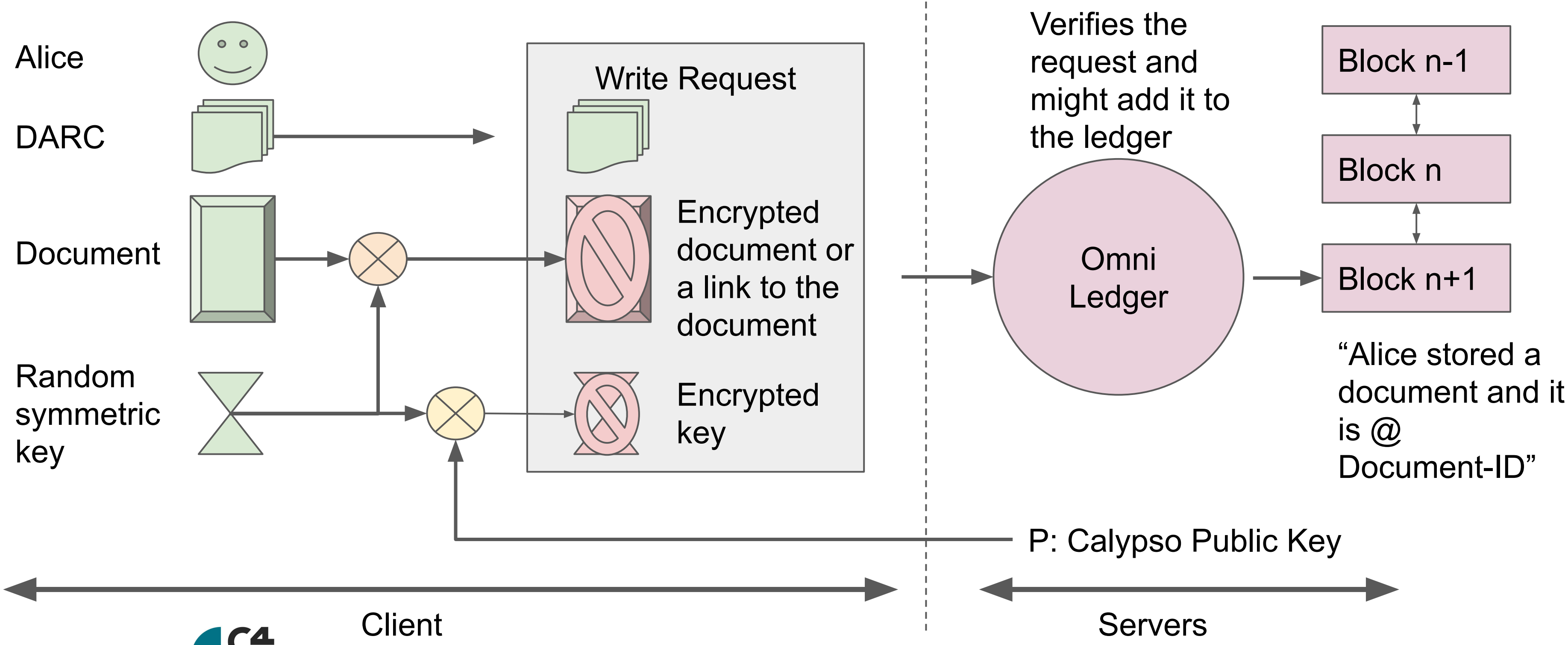
- Using homomorphic properties and zero-knowledge proofs to
  - proof the re-encryption has been done correctly
  - add the reader-encryption in every step
- Limitation using ed25519: 31 bytes of data!

$$\begin{tikzcd}
\mathrm{Enc}_{\mathrm{P1+P2+P3}}(\mathrm{Data}) \arrow[r, "\boxed{1}"] & \mathrm{Enc}_{\mathrm{P2+P3+Reader}}(\mathrm{Data}) \arrow[r, "\boxed{2}"] & \mathrm{Enc}_{\mathrm{P3+Reader}}(\mathrm{Data}) \arrow[r, "\boxed{3}"] & \mathrm{Enc}_{\mathrm{Reader}}(\mathrm{Data})
\end{tikzcd}$$

IIb. DARC usage

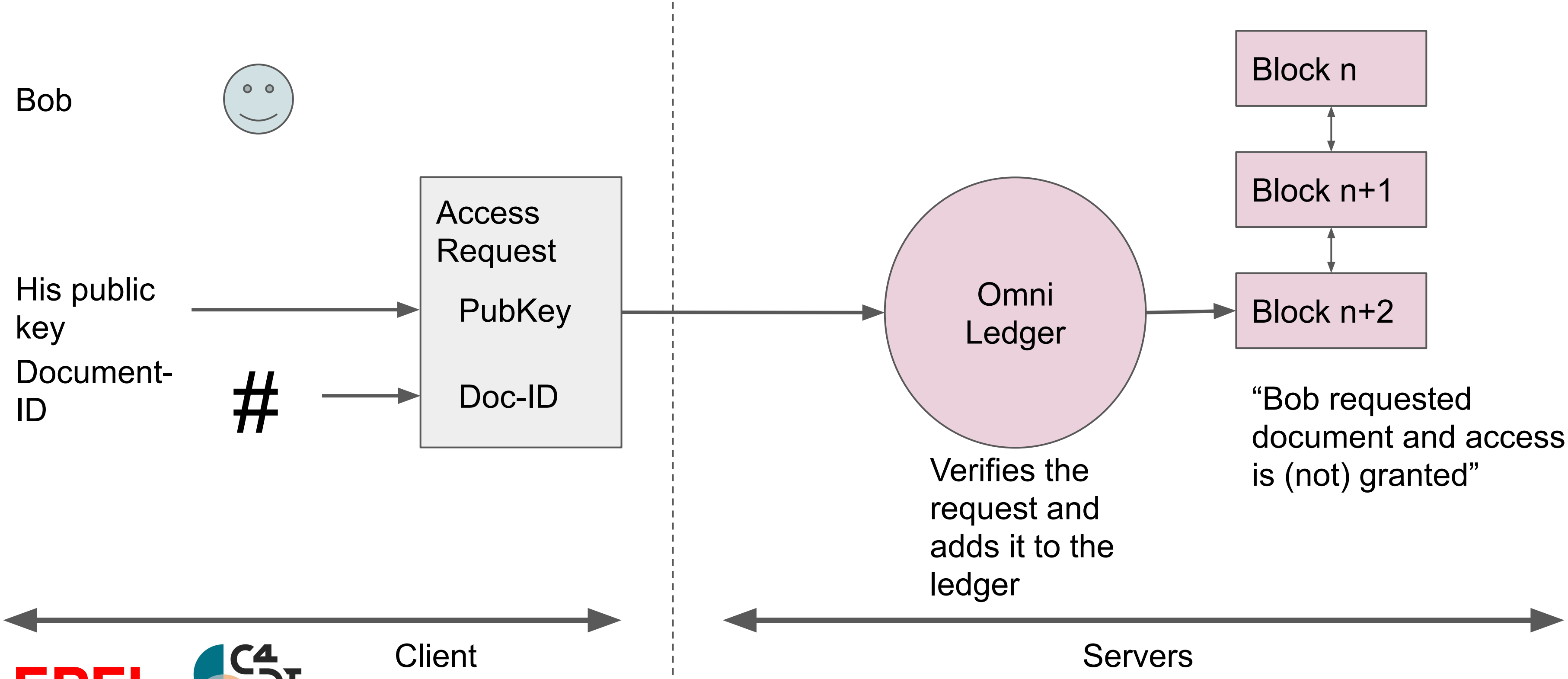
- How to
  - proof the reader is allowed to re-encrypt?
  - change access rights after encryption?
  - -> describe the access rights using DARCs
- How to avoid fraudulent re-encryption requests?
  - Writer needs to proof he knows the clear-text secret
  - -> more zero knowledge proofs

IIb. Alice Encrypts her Document


Alice
DARC
Document
Random symmetric key
Write Request
Encrypted document or a link to the document
Encrypted key
Verifies the request and might add it to the ledger
Omni Ledger
Block n-1
Block n
Block n+1
“Alice stored a document and it is @ Document-ID”
P: Calypso Public Key
Client
Servers

© 2020 by c4dt.org - Linus.Gasser@epfl.ch

IIb. Bob Requests Reading the Document



Bob

His public key

Document-ID

#

Access Request

PubKey

Doc-ID

Omni Ledger

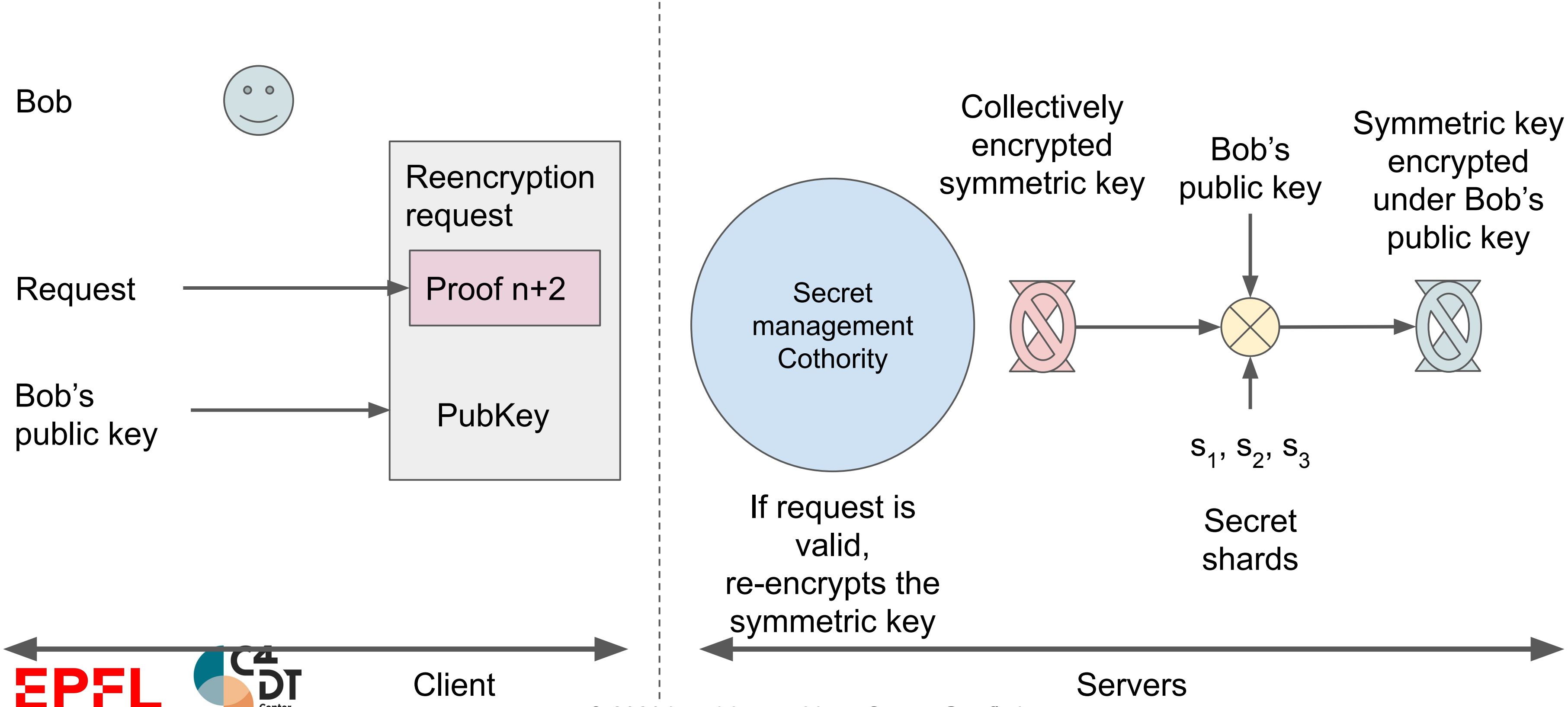
Verifies the request and adds it to the ledger

“Bob requested document and access is (not) granted”

Client

Servers

IIb. Bob Requests the Document’s Symmetric Key



Bob

Request

Bob’s public key

Reencryption request

Proof n+2

PubKey

Secret management Cothority

If request is valid, re-encrypts the symmetric key

Collectively encrypted symmetric key

Bob’s public key

Symmetric key encrypted under Bob’s public key

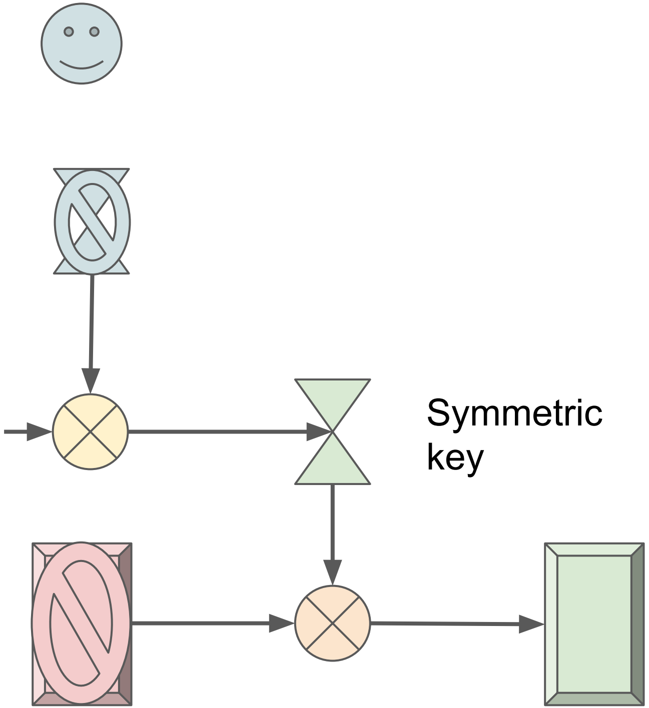
s_1, s_2, s_3

Secret shards

Client

Servers

IIb. Bob Decrypts the Document



Bob

Encrypted symmetric key

His private key

Symmetric key

Encrypted document

Decrypted document

IIb. Summary

- Alice
  - Doesn’t need to know the reader in advance
  - As long as a majority of the nodes are honest, nobody can read the message
  - Can adjust the readers over time
- Bob
  - Can trust nobody else sees the document
- Eve
  - Cannot interfere
- Auditor
  - Sees who accessed the data

IIc. Calypso Exercise

- Encrypt a text using Calypso
  - Everybody in the group uses stackblitz to *Create Calypso-Write*
  - Use your group-DARC for the access control
  - Post the WriteID to matrix
- Decrypt the texts from your group
  - Use *Read Calypso* in stackblitz
  - Find the transaction in the blockchain
    - Can you see who read the secret?
    - How could you find out?
- Try to decrypt texts from another group
  - Take a WriteID from matrix from another group and try to decrypt it
  - Find the failed transaction in the blockchain

IId. Bonus

- How could you avoid giving away who read the secret?

IIIa. Auditing using Calypso

- Understand how to support off-chain encryption
- Exercise on-chain

IIIb. Auditing Encrypted Certificates

Given product info from Pharma

Attach delegation for audits

Different auditors for different batches

Multi-signatures
Time-based
Location-based

IIIb. Auditing

- Application of Calypso
- System encrypts logs to Calypso public key
- Adds DARC for allowing re-encryption
- When auditors come, delegate re-encryption to them

IIIc. Auditing Exercise

- In each group, discuss what DARCs can be used to implement this use-case
  - 1 - every factory is independant
  - 2 - “one DARC to rule them all”
  - What are the security issues for both solutions?
  - Chose one solution and create the DARCs, share the IDs in your group
- Create the Exercise:
  - Person 1 - auditor
  - Person 2..n - factories
  - Factories create *CalypsoWrite* with the DARC chosen in the previous discussion
  - After the CalypsoWrite is done, every factory adds the auditor to its DARC
  - The factories send the writeIDs to the auditor
  - The auditor checks the messages

IIId. Bonus

- Switch the auditor to one of the other group, and let it decrypt all messages