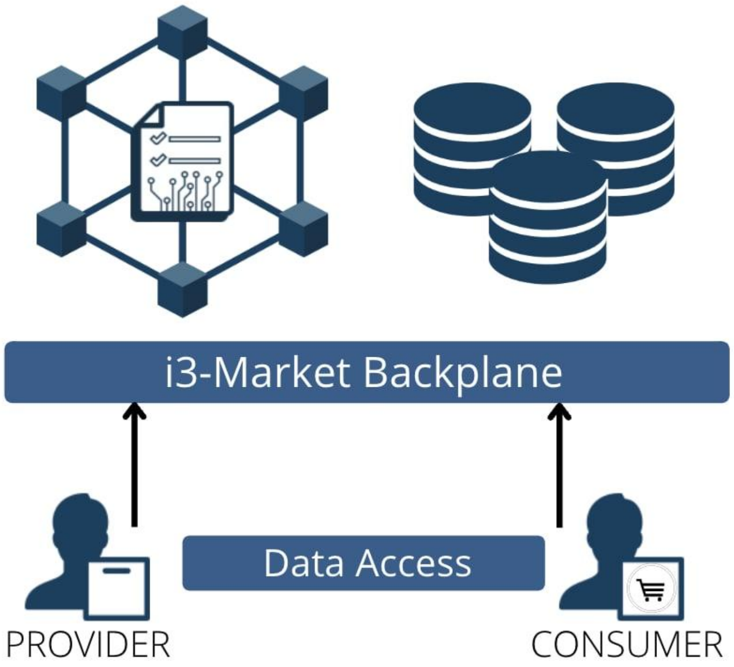
Data Marketplaces with a Free Sampling Service

Rafael Genés-Durán, Oscar Esparza, Juan Hernández-Serrano, Fernando Román-García, Miquel Soriano and Jose L. Muñoz-Tapia

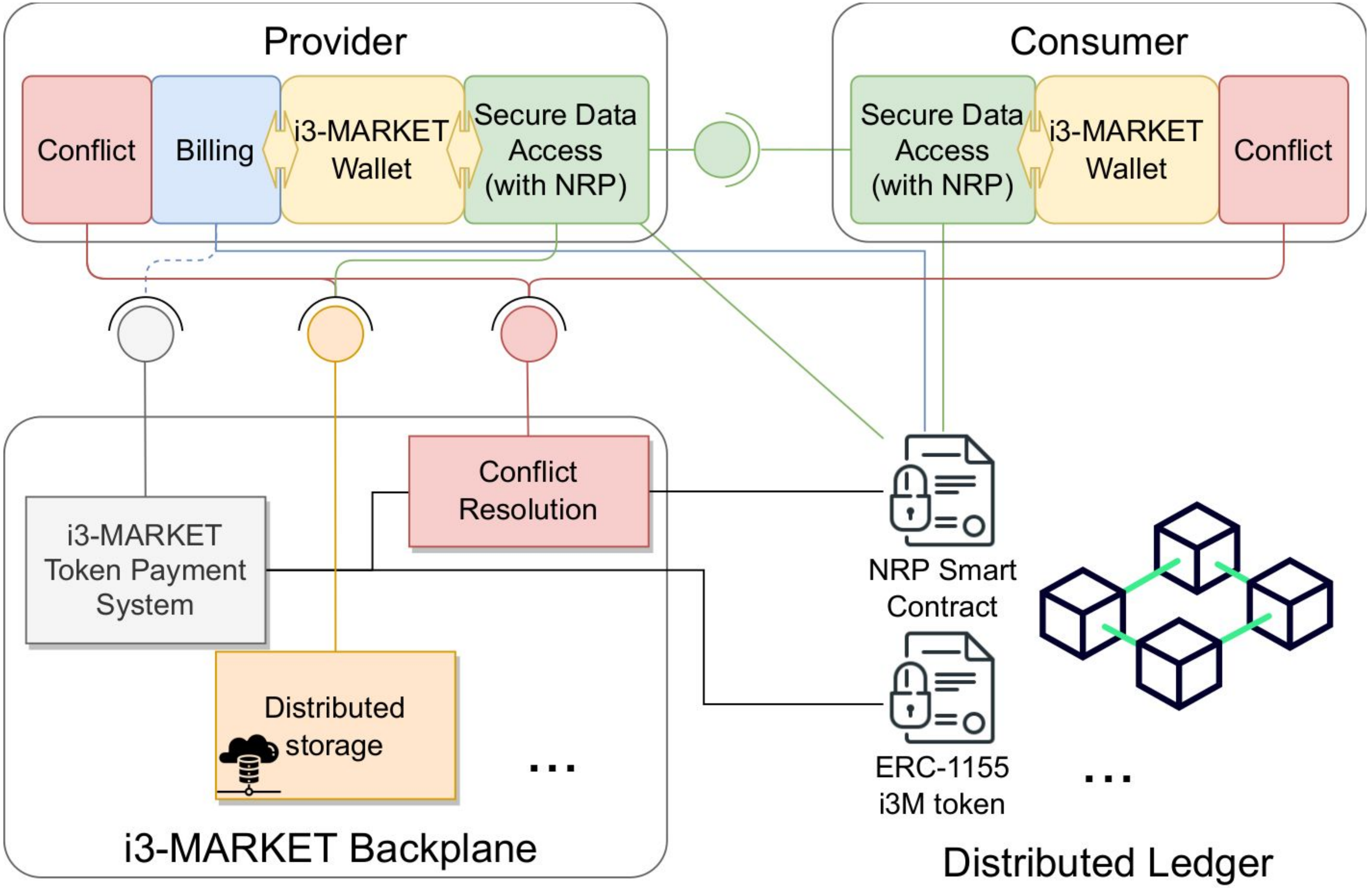
Universitat Politècnica de Catalunya, Barcelona, España

H2020 — i3-MARKET

- intelligent
- interoperable
- integrative
- deployable **MARKET**place platform



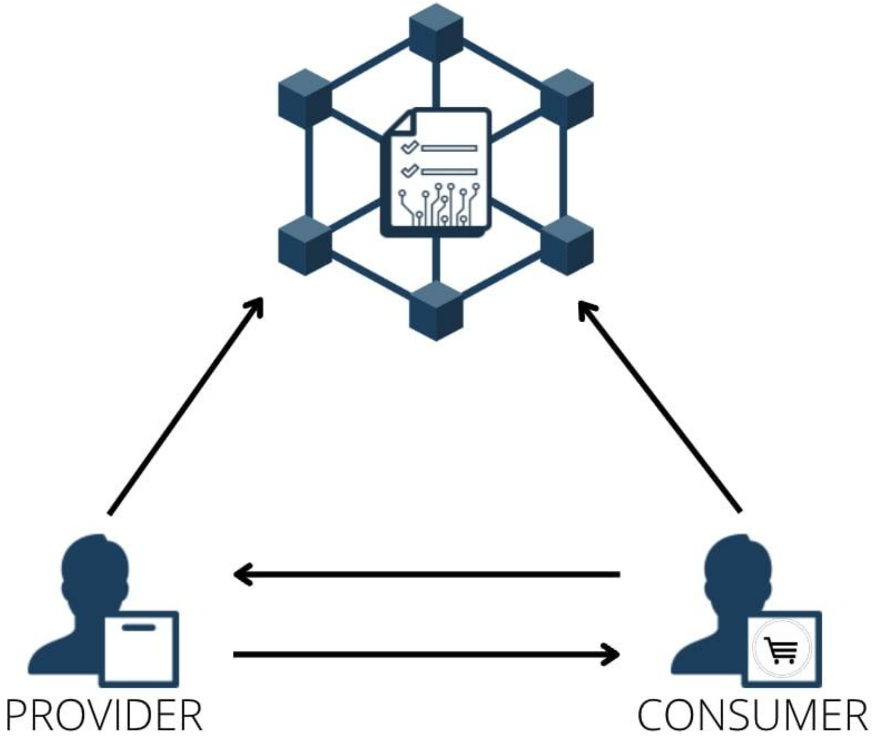
i3-MARKET components



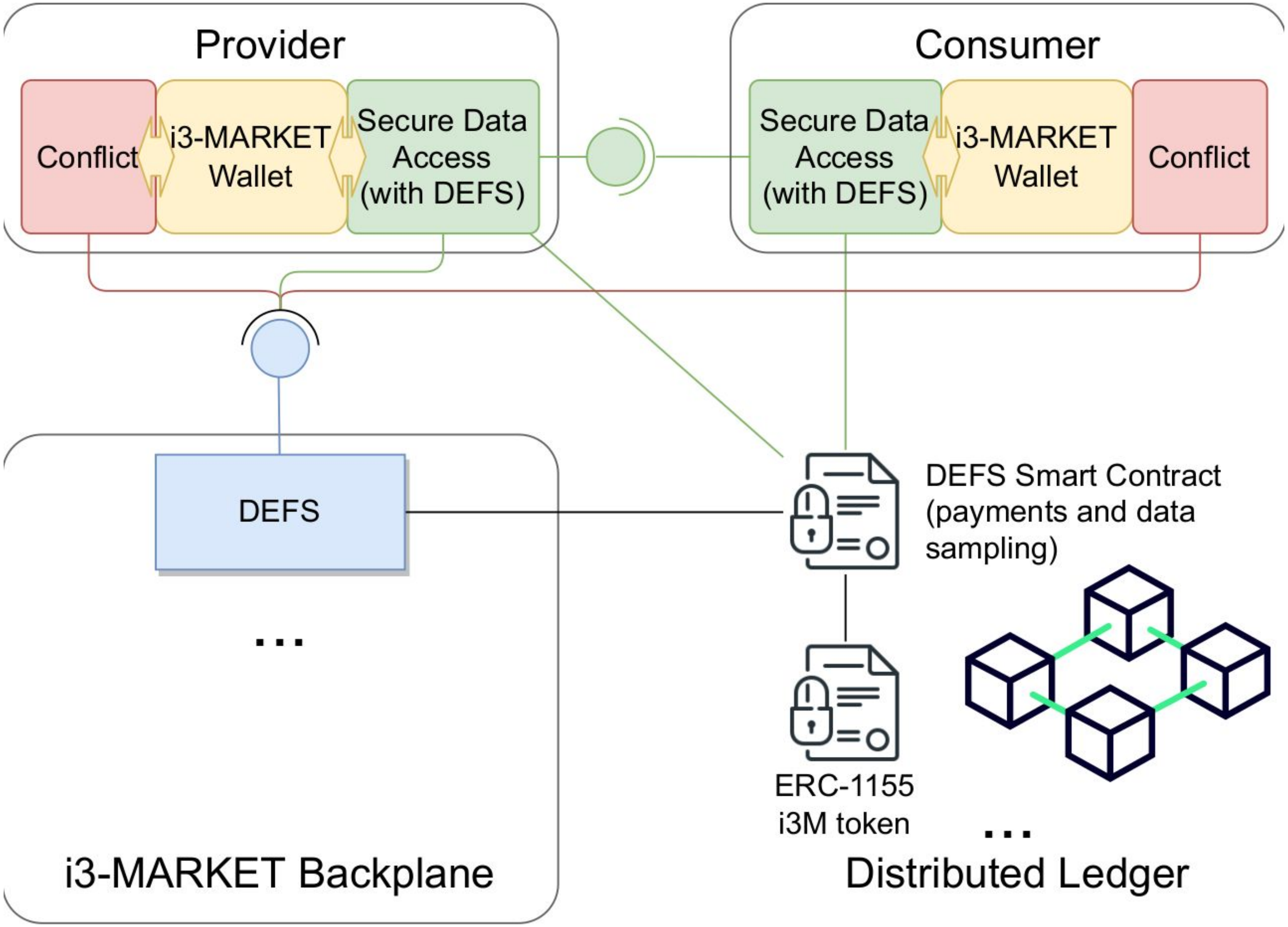
Sampling Service: DEFS

Data Exchange with Free Sampling protocol.

1. Data sample evaluation
2. Payment guarantees
3. Cost-efficiently
4. Non-repudiation
5. Liveness

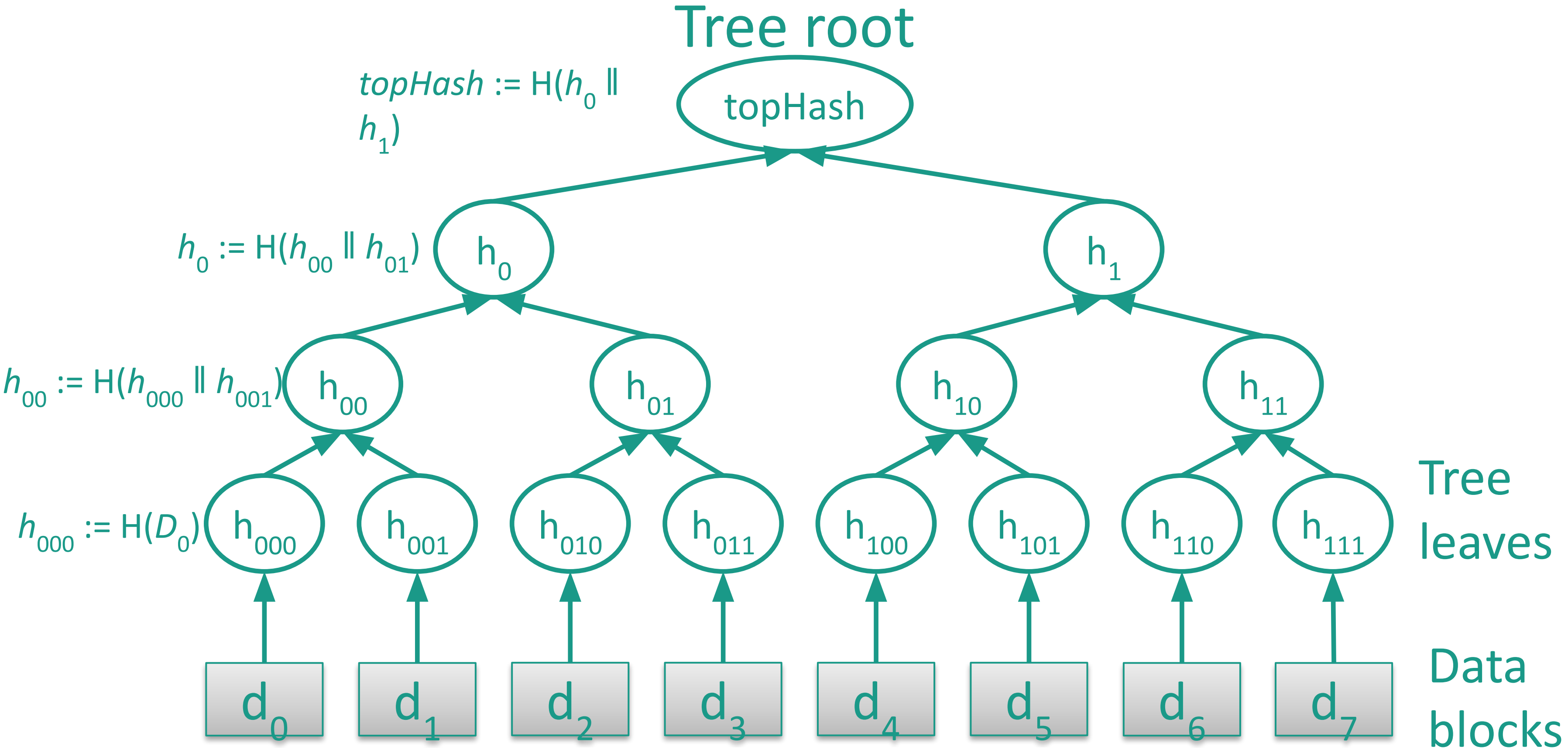


Integration of DEFS in i3-MARKET



Background: Merkle Tree

Number of blocks n.
If the tree is balanced,
tree depth $= \lceil \log_2 n \rceil + 1$



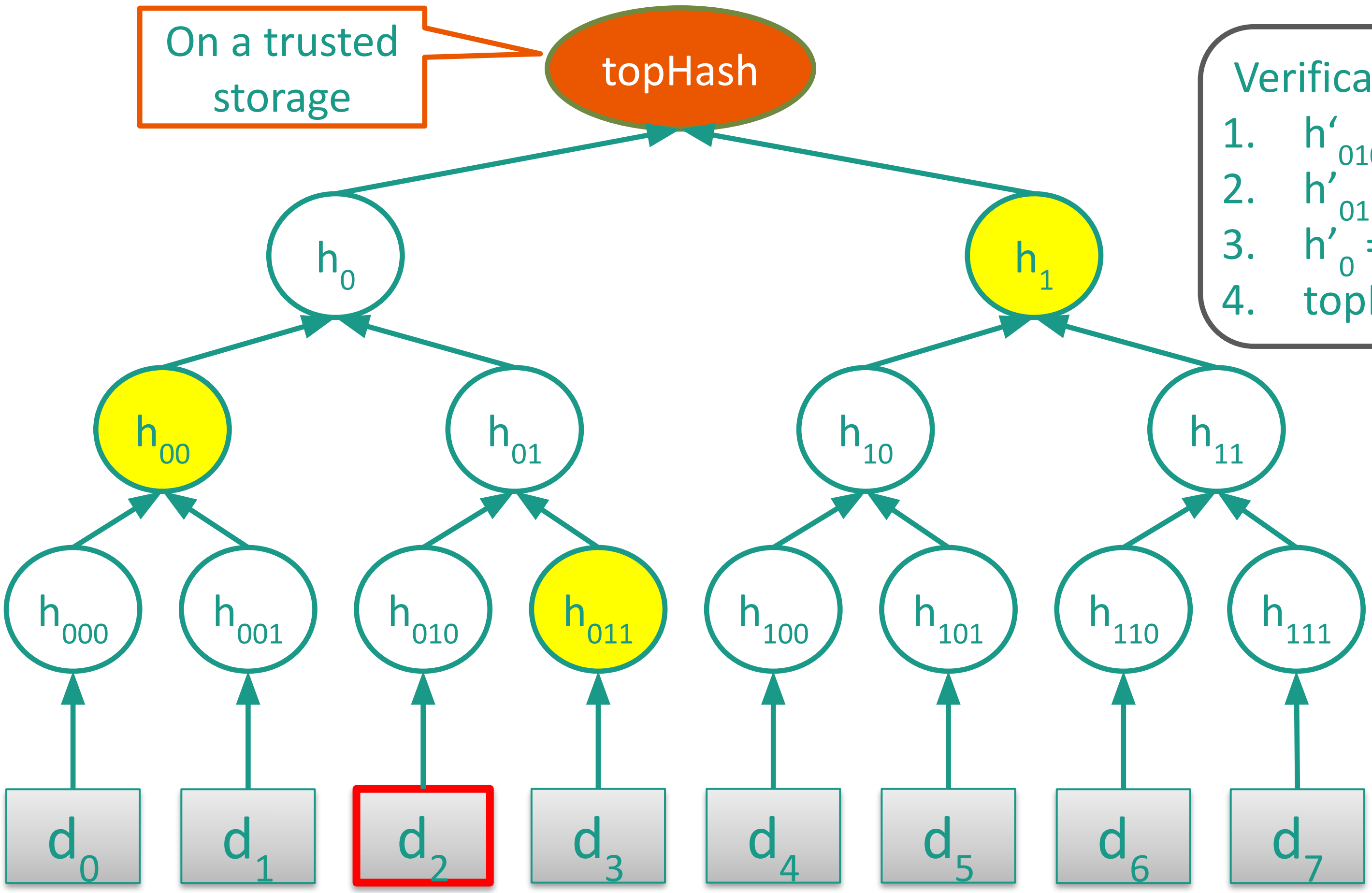
topHash := $H(h_0 \parallel h_1)$

$h_0 := H(h_{00} \parallel h_{01})$

$h_{00} := H(h_{000} \parallel h_{001})$

$h_{000} := H(D_0)$

Background: Merkle Tree



On a trusted storage

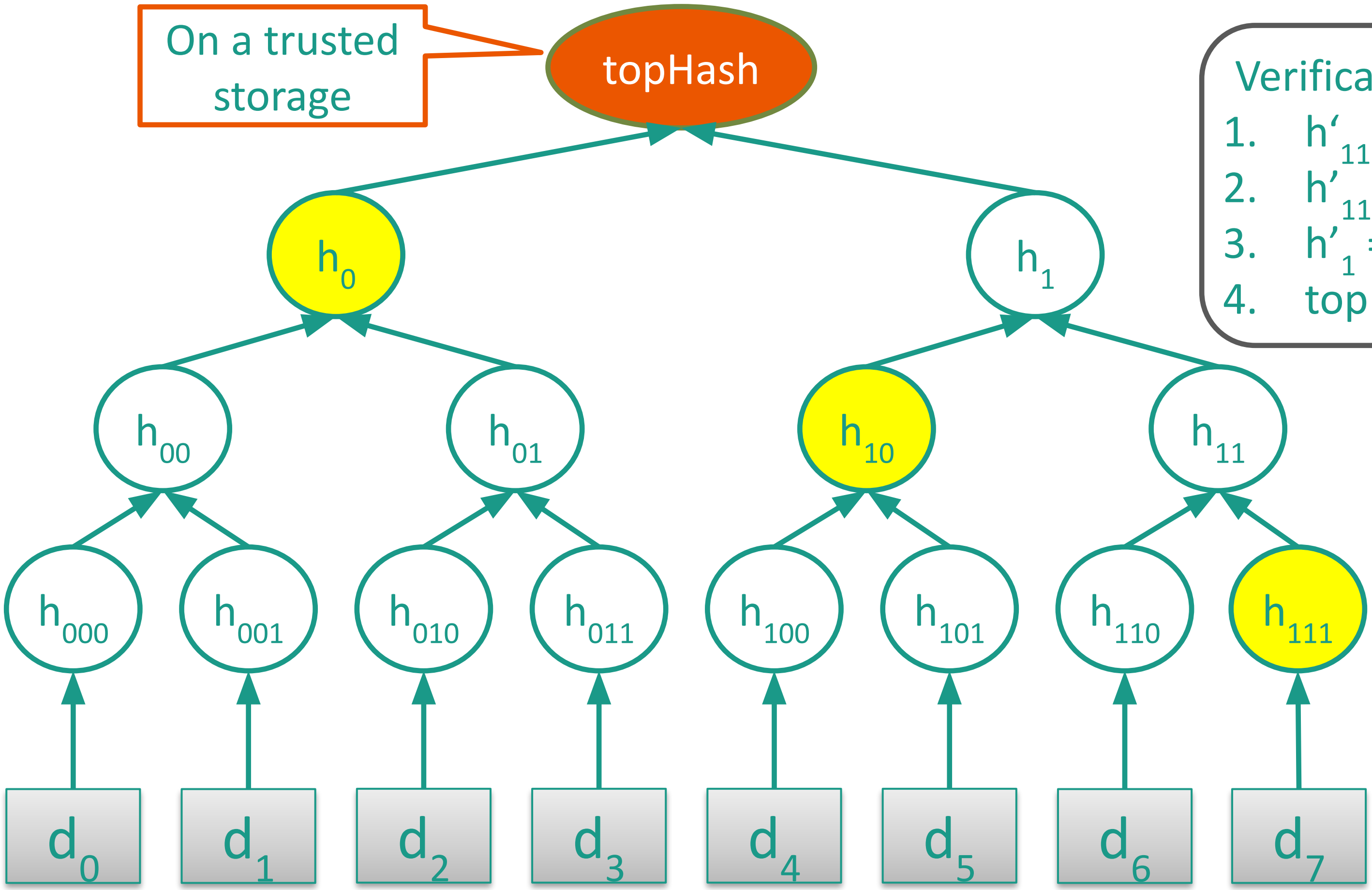
$MP_2=(h_{011}, h_{00}, h_1)$

Verification of d_2 with MP_2:

1. $h'_{010} = H(d_2)$
2. $h'_{01} = H(h'_{010} \parallel h_{011})$
3. $h'_{0} = H(h_{00} \parallel h'_{01})$
4. topHash == $H(h'_{0} \parallel h_1)$?

$O(\log_2(n))$

Background: Merkle Tree



On a trusted storage

Verification of d_6 with MP_6:

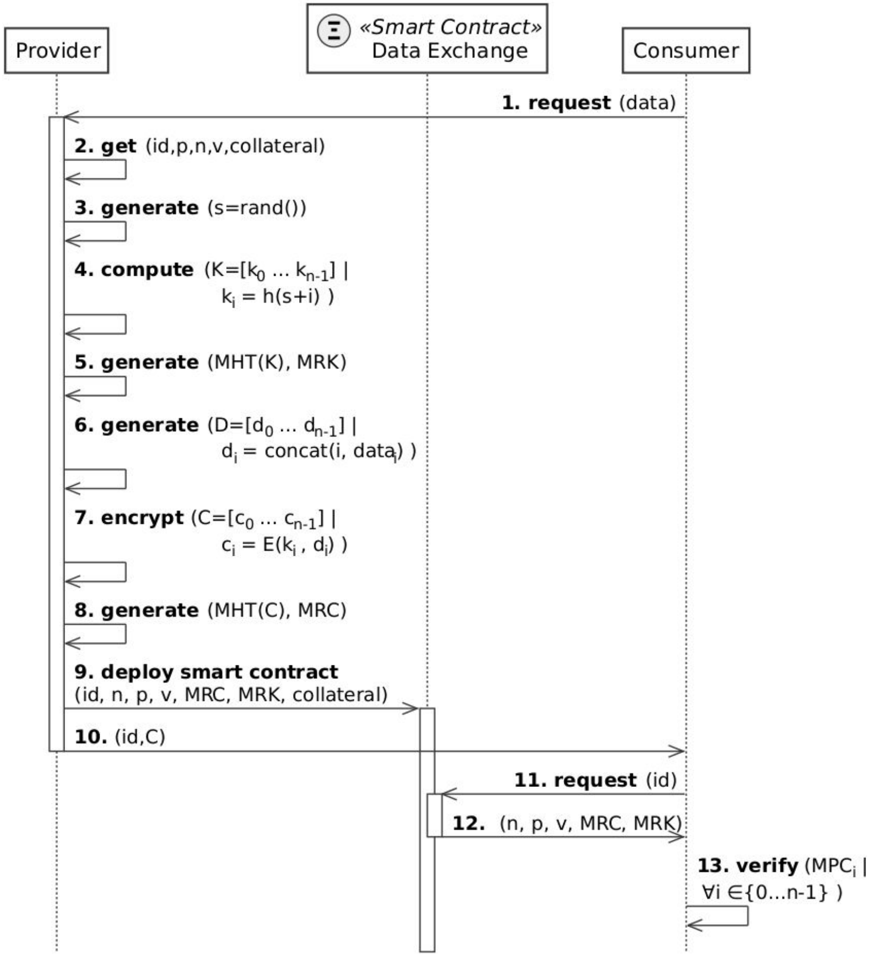
1. $h'_{110} = H(d_6)$
2. $h'_{11} = H(h'_{110} \| h_{111})$
3. $h'_{1} = H(h_{10} \| h'_{11})$
4. topHash == $H(h_0 \| h'_1)$?

$O(\log_2(n))$

$MP_6=(h_{111}, h_{10}, h_0)$

DEFS: Protocol preparation

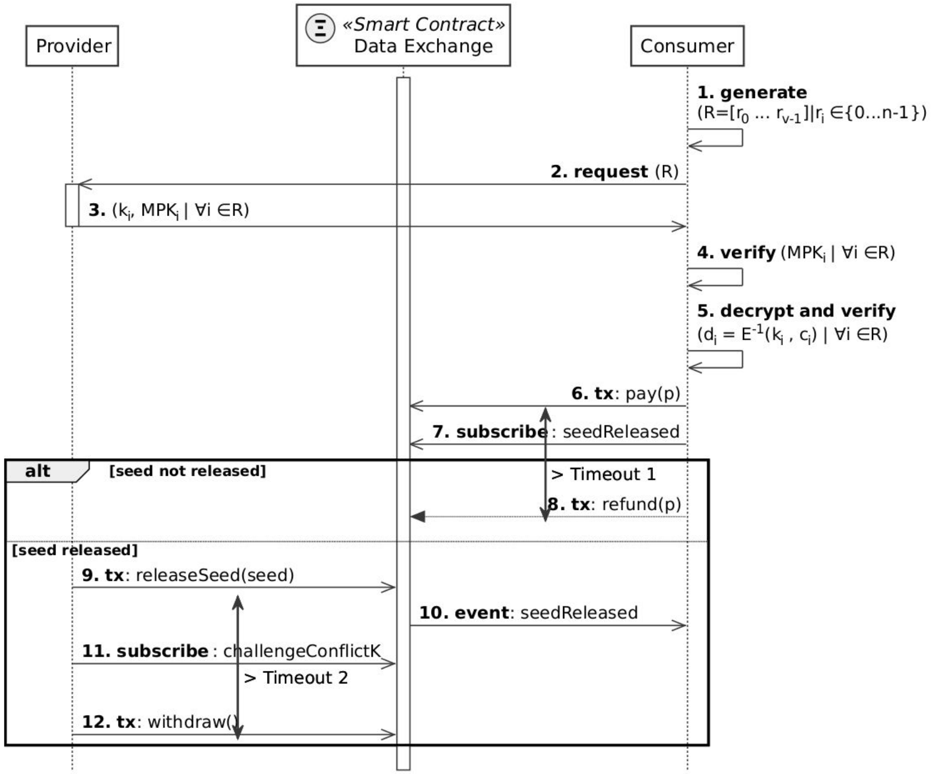
- Generates a random permutation of the dataset blocks
- Generates as many keys as blocks in the dataset based on a seed
- Create a Merkle Tree of keys
- Encrypt every block with its corresponding keys
  - ○ it creates a randomized encrypted dataset
- Creates another Merkle Tree for the encrypted dataset
- Initiates a **smart contract** with registering the merkle roots and certain parameters
- The consumer can download the encrypted blocks and check them against the merkle root publish on the SC



Provider
«Smart Contract»
Data Exchange
Consumer
1. request (data)
2. get (id,p,n,v,collateral)
3. generate (s=rand())
4. compute (K=[k_0 … k_{n-1}] |
k_i = h(s+i))
5. generate (MHT(K), MRK)
6. generate (D=[d_0 … d_{n-1}] |
d_i = concat(i, $data_i$))
7. encrypt (C=[c_0 … c_{n-1}] |
c_i = E(k_i , d_i))
8. generate (MHT(C), MRC)
9. deploy smart contract
(id, n, p, v, MRC, MRK, collateral)
10. (id,C)
11. request (id)
12. (n, p, v, MRC, MRK)
13. verify (MPC_i |
∀i ∈{0...n-1})

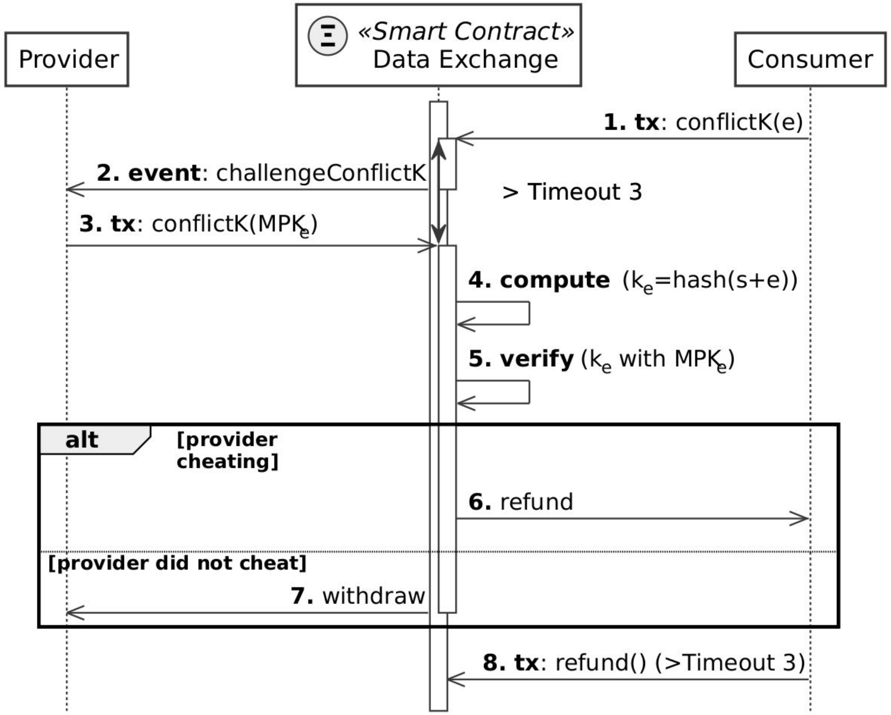
DEFS: Protocol execution

- Consumer chooses samples to be revealed
- Provider send keys to decrypt them
- Consumer checks quality
- If OK, consumer pays with tokens
    - ○ tokens are locked by the SC
- Provider publishes the seed
    - ○ consumer can decrypt the entire dataset, or starts the conflict resolution phase


Provider
«Smart Contract» Data Exchange
Consumer
1. generate $(R=[r_0 \ldots r_{v-1}] | r_i \in \{0 \ldots n-1\})$
2. request (R)
3. $(k_i, MPK_i \mid \forall i \in R)$
4. verify $(MPK_i \mid \forall i \in R)$
5. decrypt and verify $(d_i = E^{-1}(k_i , c_i) \mid \forall i \in R)$
6. tx: pay(p)
7. subscribe : seedReleased
alt
[seed not released]
> Timeout 1
8. tx: refund(p)
[seed released]
9. tx: releaseSeed(seed)
10. event: seedReleased
11. subscribe : challengeConflictK
> Timeout 2
12. tx: withdraw()

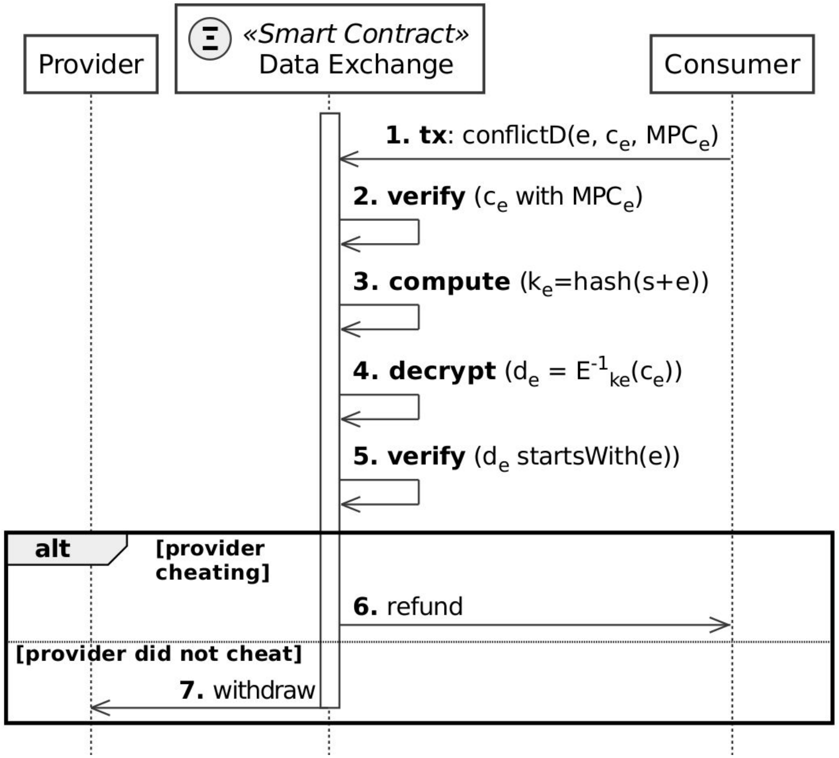
DEFS: Conflict Resolution

- Handled by the Smart Contract
- Can be started if:
  - **Keys are not properly generated.**
  - Cryptograms do not have the proper format.
- If the provider is found guilty:
  - provider pays the costs (some tokens are locked in the beginning)
  - consumer is refunded
- Otherwise:
  - consumer pays for the conflict costs
  - provider is paid


Provider
«Smart Contract» Data Exchange
Consumer
1. tx: conflictK(e)
2. event: challengeConflictK
> Timeout 3
3. tx: conflictK(MPK_e)
4. compute (k_e=hash(s+e))
5. verify (k_e with MPK_e)
alt
[provider cheating]
6. refund
[provider did not cheat]
7. withdraw
8. tx: refund() (>Timeout 3)

DEFS: Conflict Resolution

- Handled by the Smart Contract
- Can be started if:
    - Keys are not properly generated.
    - **Cryptograms do not have the proper format.**
- If the provider is found guilty:
    - provider pays the costs (some tokens are locked in the beginning)
    - consumer is refunded
- Otherwise:
    - consumer pays for the conflict costs
    - provider is paid



Provider
«Smart Contract»
Data Exchange
Consumer
1. tx: conflictD(e, c_e, MPC_e)
2. verify (c_e with MPC_e)
3. compute (k_e=hash(s+e))
4. decrypt ($d_e = E^{-1}_{ke}(c_e)$)
5. verify (d_e startsWith(e))
alt
[provider cheating]
6. refund
[provider did not cheat]
7. withdraw

Thank you!

Rafael Genes-Durán

rafael.genes@upc.edu