Non-repudiable data-exchange log **Provider** i3-Market Backplane Consumer SDA API Backplane API Backplane API : SDA SDK : Auditable Accounting : Conflict Resolution : Data Transfer Manager GET data {id: #X} create one time secret S encrypt data with S C=Enc(S, data) create proof of origin PoO=signature<sub>P</sub>({ dataDescription describes the piece of plaintext data id: #X, src: ProvId, (format, size, ...) dst: ConsId, dataCommitment is a commitment timestamp: TS<sub>0</sub>, that can be later used to dataDescription, verify the decrypted data dataCommitment: hash( data), SCommitment is a commitment SCommitment: hash(S), that can be later used to encryptedData C verify the received S {id: #X, src: ProvId, dst : ConsId, timestamp: TS<sub>0</sub>, dataDescription, dataCommitment: hash(data Scommitment: hash(S), encryptedData: C, proof: Po0} validate PoO alt [invalid PoO] terminate [valid PoO] store PoO create proof of reception PoR=signature<sub>C</sub>({ id: #X, src: ProvId, dst: ConsId, timestamp:  $TS_R$ , dataDescription, dataCommitment: hash( data), SCommitment: hash(S), encryptedData: C }) {id: #X, timestamp:  $TS_R$ , proof: PoR} validate PoR [invalid PoR] alt terminate [valid PoR] store PoR create proof of S PoS=signature<sub>P</sub>({ id: #X, src: ProvId, dst: ConsId, timestamp:  $TS_P$ , secret: S account ([ availability: ' privateStorage', permissions: {view: [ ProvId, ConsId]}, content: {id: #X, dataDescription, PoO, PoR , PoS} }, availability: ' blockchain', content: {id: #X, S} } ]) The verificationCode includes blockchain specific information proving that the  ${\tt verificationCode}$ requested data has been accounted (e.g. tx\_id, block\_id) check that blockchain registration There is a predefined/agreed max timeout ' of S is within  $P_R$  + to wait for S to be on the blockchain timeout S, PoS, verificationCode opt [S not received] [while S not received && currentTime  $< P_R + timeout$ ] loop getS {id: #X} Although the Consumer can use i3-market to connect to the Blockchain and download S, it could also use any other connection to download it since it is in the public Blockchain S, PoS, verificationCode validate PoS check hash(S) == SCommitment validate verificationCode alt [validation failed] terminate [conflict resolution] opt ref conflict resolution [validation succeed] decrypt C with S: Dec(S, C)=decryptedData validate hash( decryptedData) == dataCommitment && decryptedData meets dataDescription [validation failed] opt terminate [conflict resolution] opt ref conflict resolution : SDA SDK SDA API Backplane API Backplane API

: Conflict Resolution

: Auditable Accounting

: Data Transfer Manager