**ScryInfo protocol layer SDK API documents v0.0.5**

**Contents**

[Scry protocol layer process 1](#_bookmark0)

[API 1](#_bookmark1)

[Sdk 1](#_bookmark1)

[ScryClient 1](#_bookmark2)

[Contract 2](#_bookmark3)

Key manager 4

Event 4

[Routine 5](#_TOC_250000)

First step to deploy SDK: SDK initialization 5

Second step: Create general account, transfer Eth and token from super account to this account for transaction 5

Now we can start transaction 6

[Sellers publish data 6](#_bookmark4)

Buyers get data publish notification and description ID for this data, then they can download data description info from IPFS 7

Buyers are interested in this data after seeing description info,then they set mortagage amount and admit smart contract transfer

Token from their account for mortagage 7

Buyers are ready to buy it, smart contract deducts mortagage from buyers accoun[t 7](#_bookmark5)

Buyers get data verification ID and they can download data verification results from IPFS with this ID to judge whether this data is exactly

what they want 8

[Buyers decide to buy data 8](#_bookmark6)

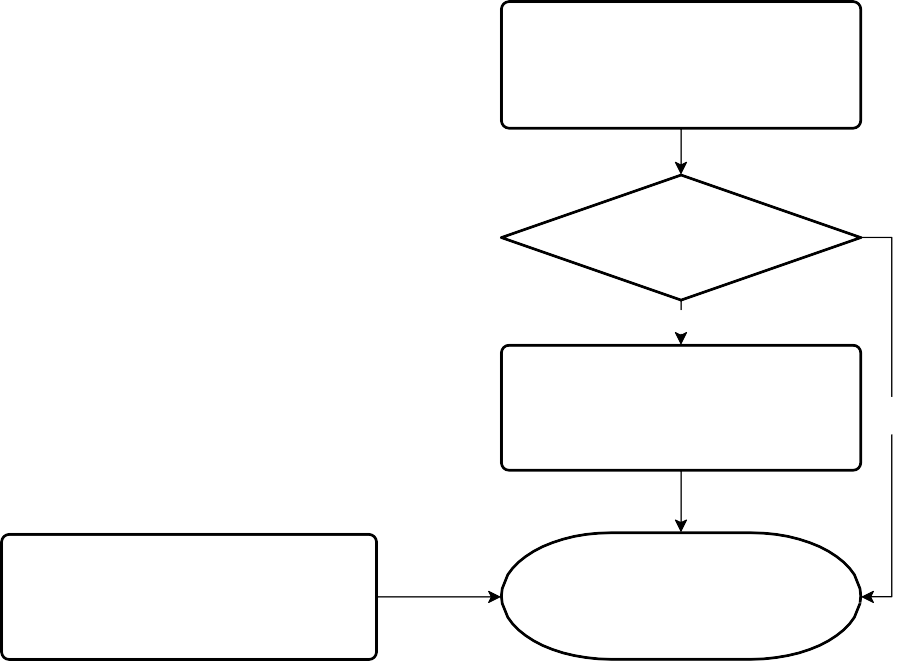
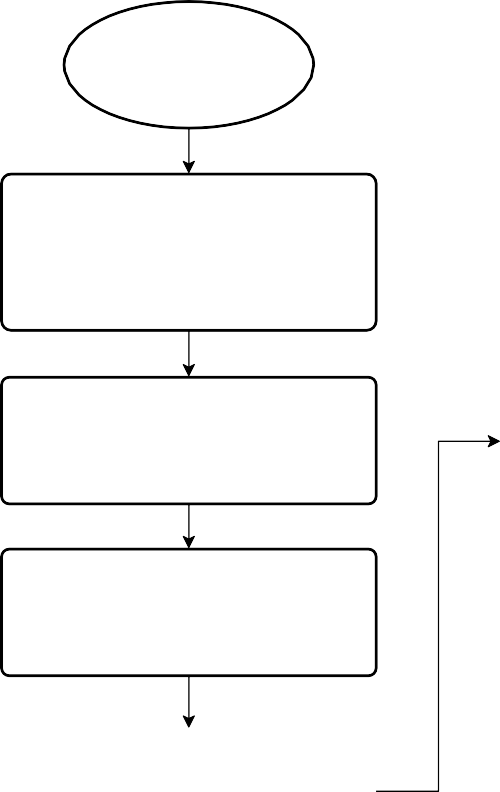
Sellers get notifications of data transaction and generate meta data id encrypted with buyers public key, then send it to contract 8

Buyers get meta data id and can download full data from IPFS 9

Buyers give feedbacks about data authenticatio[n 9](#_bookmark7)

[Transaction end notification 10](#_bookmark8)

# Scry protocol layer process



Start

Seller publishs data: me ta data, despcription data, proofs o f data.

Proofs o f data are data frag me nts whic h aims to pro ve the

truth o f data.

Buye r pre pare s to purc hase

Buyer download the c omple te me ta data

Buye r transfe rs to ke ns to c o ntrac t fo r paying to se lle r, and do wnlo ad the pro o f data(data frag me nts)

.

Y

Buye r thinks the me ta data is fake ?

N

Buye r ne e d to de c ide : buy o r no t?

Co ntrac t pay to ke ns to se lle r. Re turn buye r his de po sit.

Y

N

Co ntrac t re turn to ke ns bac k to buye r

Clo se Transac tio n.

End

## API

Sdk

Package: sdk

|  |  |  |
| --- | --- | --- |
| **seq** | **definition** | **functionality** |
| 1 | Init( | Before using scryinfo protocol layer SDK you need to initialize all SDK  et hNodeAddr : Eth node address provided by sponsorkeyServiceAddr: Key service address provided bysponsor  protocolAddr: Protocol contract address provided by sponsor  tokenAddr: token address provided by sponsor fromBlock: event scan origin block, default setting is 0 ipfsNodeAddr : IPFSnodeaddress providedbysponsor |
|  | ethNodeAddr string, keyServiceAddr string, protocolAddr string, tokenAddr string, fromBlock uint64, ipfsNodeAddr string)  (error) |

ScryClient

Package: sdk/scryclient

|  |  |  |
| --- | --- | --- |
| **seq** | **definition** | **functionality** |
| 1 | CreateScryClient( password string  ) (\*ScryClient, error) | After API deployment success,a ScryClient will be created(such as named scryClient), including public&private key account or creator, private key will be stored after encryption  User can get account address after visiting scryClient .Account .Address password: The password of account to be created, please keep carefully |
| 2 | SubscribeEvent( eventName string, callback  chainevents.EventCallback) | ScryClient can order certain event on chain by deploying this API When event comes it will trigger certain callback function. Event format can be ordered is: |
|  | DataPublish TransactionCreate Buy ReadyForDownload TransactionClose Approval |
| 3 | Authenticate( password string  ) (bool, error) | Verify user password is correct or not password: Password to be verified |
| 4 | TransferEthFrom(  from common.Address, password string, value  \*big.Int  ) (error) | Transfer ETH from address called “from” to this client  f rom: Origin account  password: Origin account password  value: Amount to be transferred, unit: 1 wei |
| 5 | TransferTokenFrom(  from common.Address, password string, value  \*big.Int  ) (error) | Transfer Token from address called “from” to this client  f rom: Origin account  password: origin account password, used for transaction signature value: Amount to be transferred, unit: 1 token |
| 6 | GetEth(  owner common.Address  ) (\*big.Int, error) | Get ETH amount from account address called owner, unit: 1 wei owner: account address |
| 7 | GetScryToken(  owner common.Address  ) (\*big.Int, error) | Get Token amount from from account address called owner, unit:1 token owner: account address |

Contract

Package: sdk/scryclient /chainint erf acewrapper

|  |  |  |
| --- | --- | --- |
| **seq** | **definition** | **functionality** |

|  |  |  |
| --- | --- | --- |
| **seq** | **definition** | **functionality** |
| 1 | Publish(  txParams  \*op.TransactParams, price  \*big.Int, metaData []byte, proofDatas [][]byte, proofNum int, descriptionData []byte  ) (string, error) | Publish data to IPFS and store source id returned from IPFS in blockchain contracts  The first return value from this API is publishId，type is string。txParams: blockchain transaction parameters  price: data price, unit: 1 t oken metaData: Da t a to be so l d  proof Datas: proof of data piece to verify data authentication  proof Num: Array size of proof of data piece  descript ionData : Data description, such as title, keyword etc. Specified meaning should be decided by application layer |
|  |  | When all data is published to IPFS and blockchain , it will send DataPublish event to all users.  DataPublish event is transmitted with Json format, please refer to Event API |
| 2 | ApproveTransfer( txParams  \*op.TransactParams, spender common.Address, value \*big.Int  ) (error) | Admit address spender transfer token from deployment address of this API |
|  | txParams: Blockchain transaction parameters  spender: Transfer admitted origin address |
|  | value: Transfer amount admitted, unit:1 token, should be higher than/equal to data to be purchased |
|  | In this process, chain will send Approval event to deployer after getting request,please refer to Event API |
| 3 | PrepareToBuy( txParams  \*op.TransactParams, publishId string  ) (error) | Buyers are ready to buy data and hope to get data verification ID Before deploying this API, buyers should deploy ApproveTransfer API  first and admit contract transfer mortagage from buyers address  txParams:Blockchain transaction parameter publishId: Same as above  Chain will send TransactionCreate event to buyers after getting request TransactionCreate event is transmitted with Json format, please refer to Event API |
| 4 | BuyData(  txParams  \*op.TransactParams, txId  \*big.Int  ) (error) | Buyers buy data  txParams: Blockchain transaction parameters  txId: Transaction ID this time, obtained from transactionId array in TransactionCreate event  Chain will sent Buy event to sellers after getting request  Buy event is transmitted with Json format, please refer to Event API |
| 5 | SubmitMetaDataIdEncWithBuy er( txParams  \*op.TransactParams, txId  \*big.Int, encyptedMetaDataId  []byte  ) (error) | Sellers upload metaDataId of data to be purchased, this ID is encrypted with buyers public key  txParams: Blockchain transaction parameters  txId: transaction ID this time, obtained from transactionId array in Buy event  encyptedMetaDataId : metaDataId encrypted with buyers public key Chain will send ReadyForDownload event to buyers after getting request Buyers get metaDataId and decrypted with private key, then get the real  met a dat a ID，can download data from IPFS  ReadyForDownload event is transmitted with Json format, please refer to  Event API |
| 6 | ConfirmDataTruth( txParams  \*op.TransactParams, txId  \*big.Int, truth bool  ) (error) | Buyers confirm data authentication |
|  | txParams: Blockchain transaction parameters  txId: Transaction ID this time, obtained from transactionId in Buy event  truth: Whether the data is ture or not |

|  |  |  |
| --- | --- | --- |
|  |  | Chain will send TransactionClose event to all users after getting the request and then end the transaction  If the data is true, contract will send the mortagage from buyers to sellers TransactionClose event is transmitted with Json format, please refer to  Event API |

Key Manager

Package: sdk/util/accounts

|  |  |  |
| --- | --- | --- |
| **seq** | **definition** | **functionality** |
| 1 | Encrypt(  plainText []byte, address string) ([]byte, error) | Encrypt plaintext with the public key of user address plainText: plaintext data  Address : user address |
| 2 | Decrypt(  cipherText []byte, address string, password string)  ([]byte, error) | Decrypt the ciphertext with the private key of user’s address  cipherText : cipher text address: user address password : user password |
| 3 | ReEncrypt(  cipherText []byte, address1 string, address2 string, password string) ([]byte, error) | The encrypted data cipherText is decrypted and then encrypted using the public key of the second user address.  cipherText: ciphertext data address1 : decrypt user address address2 : encrypted user address password: decrypt user password |

Event

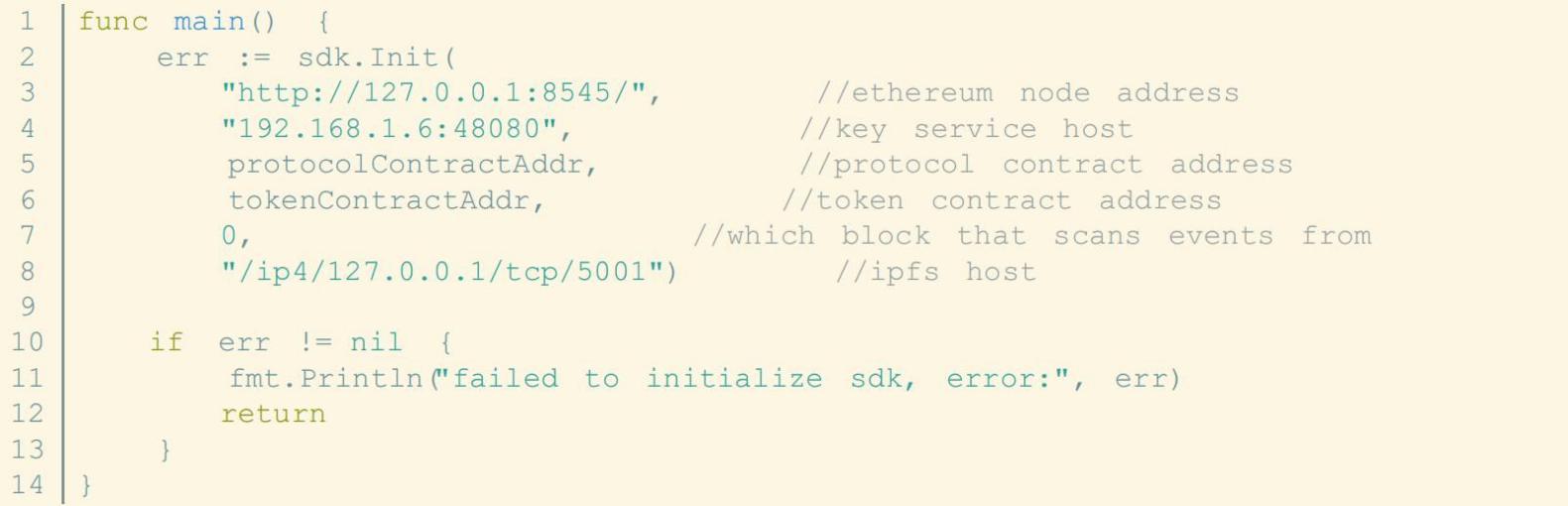
All events are transmitted in JSON format.

|  |  |  |
| --- | --- | --- |
| seq | event name | example |
| 1 | DataPublish | {  "despDataId":"QmhKnroYBFp", // Data description ID, based on wich the description of the data for sale can be obtained from IPFS  "price":1000, //Price for sale  "publishId":"155212056", //the released ID is the same as the first return value of Publish "seqNo":"155212057",  "users":["0xxxx"]  } |
| 2 | Approval | {  "owner":"0x3Ab0dAA324", //authorized person  "spender":"0x3c4d26E91",// authorized pperson"value":1600 //has been granted  } |
| 3 | TransactionCreate | {  "proof Ids":[[211,159],[170,49]], //data proof ID "publishId":"1552121", "transactionId":4, //transaction ID  "users":["0xxxx"]  } |

|  |  |  |
| --- | --- | --- |
| seq | event name | example |
| 4 | Buy | { |
|  |  | "metaDataIdEncSeller":"SdWeISqpSyhGA==", //Metadata ID encrypted with the |
|  |  | seller’s public key |
|  |  | "publishId":"1552121827295982593-913139", //Data release ID |
|  |  | "seqNo":"1552121924035218179-67679122", |
|  |  | "transactionId":4, //Transaction ID "users":["0xxxx"] |
|  |  | } |
| 5 | ReadyForDownload | {  "metaDataIdEncBuyer":"tql/mAZY/z0aR2g==", //Metadata ID encrypted with the buyer’s public key  "seqNo":"1552126832692923243-89400636",  "transactionId":4, //Transaction ID "users":["0xxxx"]  } |
| 6 | TransactionClose | { |
|  |  | "seqNo":"1552126873209063970-5512851552272651527", |
|  |  | "transactionId":4, //Transaction ID "users":["0xxxx"] |
|  |  | } |

# Routine

First step to deploy SDK: SDK initialization



Second step: Create general account, transfer Eth and token from super account to this account for transaction

The system has a built-in super account. The account address and password are shown in the example code Normal accounts need to consume gas and token for trading, so users must first transfer from super account to ETH and token

1

2

3

4

5

6

7

8

9

10

//user password

clientPassword := "888888"

//super account address, this is built into the system, the address is fixed

suAddress := "0xd280b60c38bc8db9d309fa5a540ffec499f0a3e8"

//super account address, this is built into the system, the password is fixed

suPassword := "111111" func

CreateClientWithETHToken( toke n \*big.Int,

|  |  |  |
| --- | --- | --- |
| 11  12 | client, err := scryclient.CreateScryClient(clientPassword if err != nil { | |
| 13 |  | fmt.Println("failed to create user, error:", err) |
| 14 |  | return nil, err |
| 15 |  | } |
| 16 |  |  |
| 17 |  | //Transfer into ETH |
| 18 |  | \_, err = cif.TransferEth( |
| 19 |  | common.HexToAddress(suAddress), |
| 20 |  | suPassword, |
| 21 |  | common.HexToAddress(client.Account.Address), |
| 22 |  | eth) |
| 23 |  | if err != nil { |
| 24 |  | fmt.Println("failed to transfer Eth, error:", err) |
| 25 |  | return nil, err |
| 26 |  | } |
| 27 |  |  |
| 28 |  | //Transfer into Token |
| 29 |  | txParam := chainoperations.TransactParams{ |
| 30 |  | From: common.HexToAddress(suAddress), |
| 31 |  | Password: suPassword, |
| 32 |  | } |
| 33 |  | err = cif.TransferTokens( |
| 34 |  | &txParam, |
| 35 |  | common.HexToAddress(client.Account.Address), |
| 36 |  | token) |
| 37 |  | if err != nil { |
| 38 |  | fmt.Println("failed to transfer token, error:", err) |
| 39 |  | return nil, err |
| 40 |  | } |
| 41 |  |  |
| 42 |  | return client, nil |
| 43 | } |  |
|  |  |  |

Now we can start transaction Sellers publish data

1

2

4

5

6

7

8

9

10

11

12

13

15

16

17

18

19

20

21

22

func SellerPublishData(supportVerify bool) seller.SubscribeEvent("DataPublish", onPublish) 3

//pending metadata

metaData := []byte("pussy poppy")

{

//some metadata fragments: used to prove the authenticity of metadata proofData := [][]byte{{'4', '5', '6', '3'}, {'2', '2', '1'}}

//metadata description data despData := []byte{'7', '8', '9', '3'}

txParam := chainoperations.TransactParams{

From: common.HexToAddress(seller.Account.Address),

Password: keyPassword, 14

}

cif.Publish(

&txParam, big.NewInt(1000), metaData, proofData,

2,

despData, 23

)

24 }

Buyers get data publish notification and description ID for this data, then they can download data description info from IPFS

{

|  |  |  |
| --- | --- | --- |
| 1  2 | func onPublish(event events.Event) bool  //data release ID | |
| 3 |  | publishId = event.Data.Get("publishId").(string) |
| 4 |  | //description data ID |
| 5 |  | despDataId := event.Data.Get("despDataId").(string) |
| 6 |  | //price |
| 7 |  | price := event.Data.Get("price").(\*big.Int) |
| 8 |  | return true |
| 9 | } |  |

Buyers are interested in this data after seeing description info,then they set mortagage amount and admit smart contract transfer Token from their account for mortagage

1

2

3

4

5

6

7

8

9

10

11

func BuyerApproveTransfer()

{

//if the buyer receives the onApprovalBuyer Transfer notification, the surface approve succeeds buyer.SubscribeEvent("Approval", onApprovalBuyerTransfer)

//buyerPassword: the account password of buyer txParam := chainoperations.TransactParams{

From: common.HexToAddress(buyer.Account.Address), Password: buyerPassword,

}

12

13

14

15

16

17

18

19

//protocolContractAddr is the protocol layer contract address, the amount of the deposit must be greater than or equal to the data price, otherwise the purchase operation will fail

err := cif.ApproveTransfer(&txParam, common.HexToAddress(protocolContractAddr), big.NewInt(1000))

if

err != nil

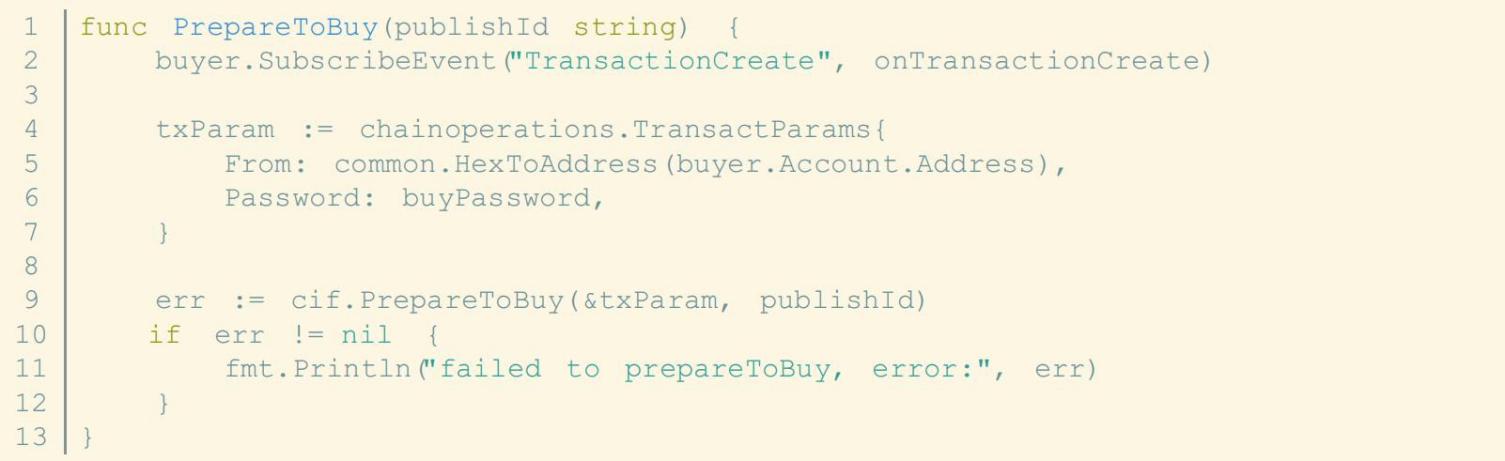
{

fmt.Println("BuyerApproveTransfer:", err)

}

}

Buyers are ready to buy it, smart contract deducts mortagage from buyers account



Buyers get data verification ID and they can download data verification results from IPFS with this ID to judge whether this data is exactly what they want

1

2

3

4

5

6

7

8

9

10

11

func onTransactionCreate(event events.Event) bool

//transaction ID

txId = event.Data.Get("transactionId").(\*big.Int)

{

//Data proof IDs

proofIDs := event.Data.Get("proofIds").([][32]byte)

return true

}

Buyers decide to buy data

1

2

3

4

5

6

func Buy(txId \*big.Int)

{

//the seller needs to order the Buy event in order to receive the buyer’s purchase

message

seller.SubscribeEvent("Buy", onPurchase)

7

8

9

10

11

12

13

14

15

16

17

//buyerPassword is the account password of the buyer txParam := chainoperations.TransactParams{

From: common.HexToAddress(buyer.Account.Address), Password: buyerPassword,

}

//txId is transaction ID

err := cif.BuyData(&txParam, txId)

if

err != nil

{

fmt.Println("failed to buyData, error:", err)

}

}

Sellers get notifications of data transaction and generate meta data id encrypted with buyers public key, then send it to contract

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

func onPurchase(event events.Event) bool

{

//meta data is encrypted with the seller’s public key

metaDataIdEncWithSeller = event.Data.Get("metaDataIdEncSeller").([]byte)

var err error

//metaDataIdEncWithBuyer： meta data is encrypted with the buyer’s public key

//metaDataIdEncWithSeller: meta data is encrypted with the seller’s public key

//sellerPassword: seller password

metaDataIdEncWithBuyer, err =

accounts.GetAMInstance().ReEncrypt( metaDataIdEncWithSeller, seller.Account.Address,

buyer.Account.Address, sellerPassword,

)

if

err != nil {

fmt.Println("failed to ReEncrypt meta data id with buyer's public key") return false

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

}

SubmitMetaDataIdEncWithBuyer(txId) return true

}

func SubmitMetaDataIdEncWithBuyer(txId \*big.Int)

//the buyer needs to monitor to the ReadyForDownload event

{

buyer.SubscribeEvent("ReadyForDownload", onReadyForDownload)

txParam := chainoperations.TransactParams{

From: common.HexToAddress(seller.Account.Address), Password: sellerPassword,

}

err := cif.SubmitMetaDataIdEncWithBuyer( &txParam,

txId, metaDataIdEncWithBuyer)

if

err != nil

{

fmt.Println("failed to SubmitMetaDataIdEncWithBuyer, error:",

err)

}

}

Buyers get meta data id and can download full data from IPFS

1 func onReadyForDownload(event events.Event) bool 2

3

4

5

6

7

8

9

10

11

{ metaDataIdEncWithBuye

//buyer to decrypt metaDataIdEncWithBuyer，get original meta data id, download the real number from IFPS based on the ID

metaDataId, err :=

accounts.GetAMInstance().Decrypt( metaDataIdEncW ithBuyer,

buyer.Account.Address,

if

err != nil {

fmt.Println("failed to decrypt meta data id with buyer's private key",

err)

12

13

14

15

16

return false

}

return true

}

Buyers give feedbacks about data authentication

1

2

3

4

5

6

7

8

9

10

11

12

func ConfirmDataTruth(txId \*big.Int)

{ buyer.Subscr

txParam := chainoperations.TransactParams{

From: common.HexToAddress(buyer.Account.Address), Password: buyerPassword,

}

//txId: transaction ID

err

:=

cif.ConfirmDataTruth( &t xParam,

|  |  |  |  |
| --- | --- | --- | --- |
| 13 |  |  | true) |
| 14 |  | if | err != nil { |
| 15 |  |  | fmt.Println("failed to ConfirmDataTruth, error:", err) |
| 16 |  | } |  |
| 17 | } |  |  |

Transaction end notification

Routine

1

2

3

func onClose(event events.Event) bool fmt.Println("onClose:", event)

return true 4 }

{