OpenCoin

a protocol for privacy preserving electronic cash payments

Version: 0.4 - draft (July 2022)

This version of the protocol build on previous work by the following authors:

Jörg Baach
Nils Toedtmann
J. K. Muennich
M. Ryden
J. Suhr

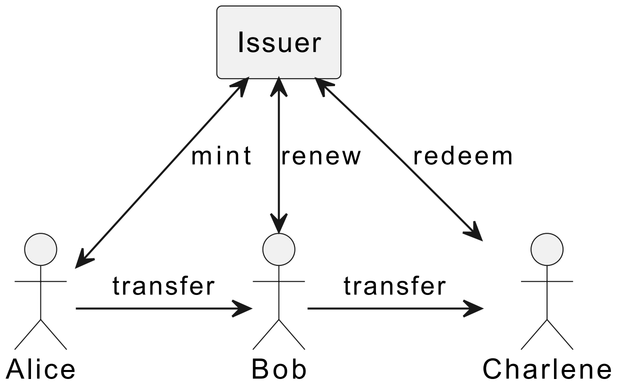
For more information go to **https://opencoin.org**

Intro

We propose a protocol that allows cash-like payments in the electronic world. It is based on the invention by David Chaum[1]. The main focus are *untraceable* payments, which means that even though there is a central entity (called the issuer, something like a bank), this central entity can't see the transactions happening. This is good for the privacy of the users.

The focus of the project is the protocol. This means we standardize the way we exchange messages and their content in order to make electronic cash payments. But we don't deliver an implementation here. That is the scope of other project(s). OpenCoin is the foundation to build upon.

How does it work?



This is a high level (but strongly simplified) image describing the basic system. We have three participants: Alice and Bob are normal users, while the Issuer is something like a bank, capable of minting coins. It also acts as an exchange for 'real-world' currency. At this high level it works as follows:

1. Alice asks the Issuer to **mint** coins. This is done in a special way using *blind signatures*, which means that the coins *can't be linked to her* later on.
2. Alice then **transfers** the coins to Bob. She can do that any way she wants, e.g. using WhatsApp, Email or any other system of her choice (also depending on what her client software supports). This could even be done by printing the coins and handing them over.
3. Bob then **renews** the coins. He swaps the coins he got from Alice for fresh coins. This way he protects himself against Alice "accidentally" using the coins

somewhere else. One can spend opencoins only once, so *double spending* needs to be ruled out, and this is done by immediately renewing received coins.
4. Bob might **transfer** the coins to yet another person, Charlene
5. Charlene decides to **redeem** the coins, meaning she asks the issuer to swap the opencoins for real-world money.

On **blind signatures**: at the core a coin is a serial number with a signature from the mint. In order to ensure that *a coin can't be traced back to the original client* we use blind signatures.
Imagine Bob hands in a coin that Alice had minted. In order to ensure the coin can't be traced back to Alice, the issuer has to sign the serial number without seeing it. In non-technical terms Alice puts the serial number in an envelope (along with carbon copy paper), and the issuer actually signs the envelope. Because of the carbon copy paper the signature presses through onto the serial number. Alice can then open up the envelope and has a signed serial, without the issuer ever seeing it.

Who is it for?

OpenCoin (the protocol) allows the development of applications for electronic cash. So firstly OpenCoin is targeted at developers. These applications however should allow everyone to make and receive electronic payments. It still requires somebody to run the central issuer. This issuer would issue an OpenCoin based electronic money system. Because electronic money is quite regulated in Europe (and other countries), the issuer would be most likely a regulated electronic money provider or a bank. We think, that a central bank would be the best issuer, because central banks issue money anyhow. But nothing technical stops you from using OpenCoin for your private project [2].

Alternatives

Why don't just use one of the alternatives?

Bitcoin / blockchain

Bitcoin (or blockchain in the more general form) is basically the opposite of OpenCoin: transfers have to happen within the system, they are visible to everybody, there is no central instance, there is no guaranteed value you can redeem the bitcoins for.

OpenCoin on the contrary makes the transfers invisible and untraceable, and has a central instance that is able to guarantee a value if you redeem the OpenCoin.

One could say that bitcoin behaves more like gold, while OpenCoin behaves more like cash.

GNU Taler

GNU Taler is build around the same central idea as OpenCoin. It started later, and is more complete than OpenCoin. They differ in the way the take care of the renewal step and coin splitting. They also make more assumptions regarding the clients (e.g. clients having key identifying them), they have clearer roles (e.g. consumer and merchant) and by all of this hope to get around the inherent problems of untraceable transfers, e.g. tax-ability.

The trade-off seems to be that their system is harder is more complex and harder to understand. We also doubt that these complexities are necessary to reach the stated goals. We also doubt that the goals can really be reached, and also find that the system's documentation is quite hard to understand. This might be because they deliver implementations for all necessary software components, and are not really targeted at other implementations of they system.

Because of all this one could say that GNU Taler is less open to other developers.

Problems

- Tax
- Money laundering
- Blackmail and other crime

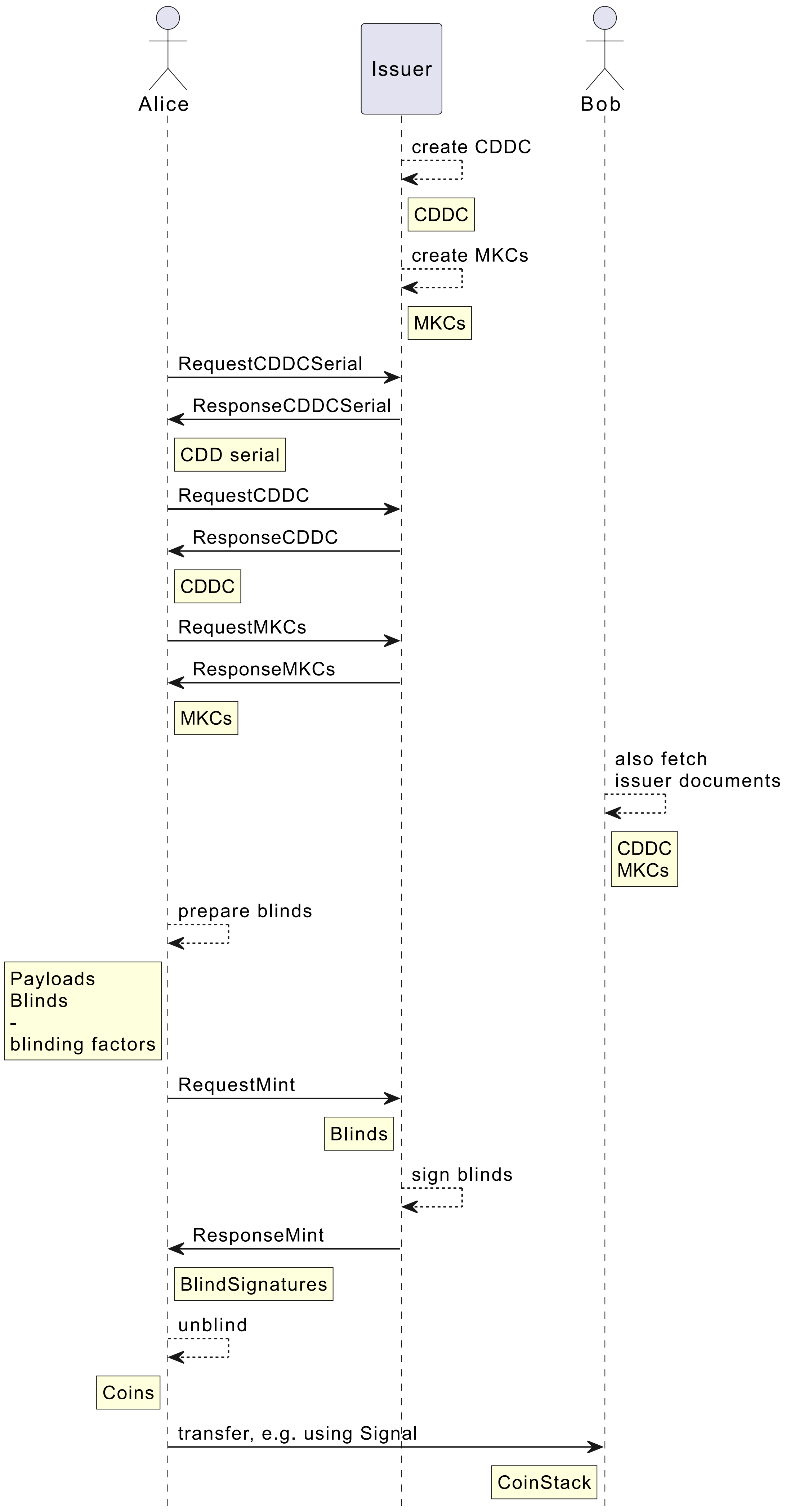
The OpenCoin protocol

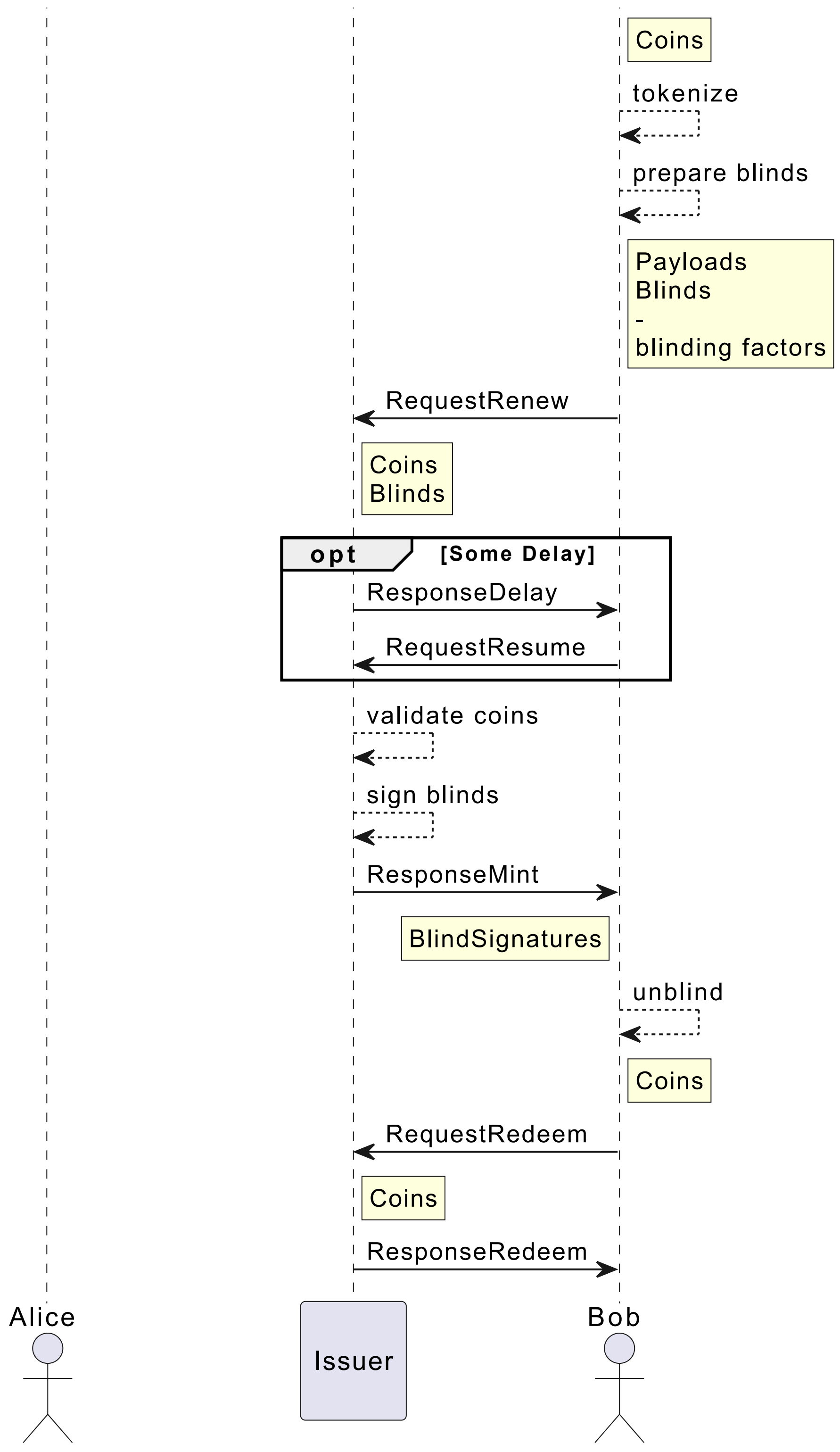
Assumptions

The exchange of messages MUST happen over a **secure channel**. For HTTPS this means TLS, but other channels like messengers provide their own. For an email exchange GPG would be recommended. Either way, it is the responsibility of the developer to take care of the transport security.

When requesting the issuer to mint or redeem coins some form of **authentication & authorization** is most likely required - the issuer needs to secure payment for the coins, or make a payment somewhere for redeemed coins. Because auth* might already be provided by the transport layer, we don't include it in the OpenCoin protocol.

Overview


Alice
Issuer
Bob
create CDDC
CDDC
create MKCs
MKCs
RequestCDDCSerial
ResponseCDDCSerial
CDD serial
RequestCDDC
ResponseCDDC
CDDC
RequestMKCs
ResponseMKCs
MKCs
also fetch
issuer documents
CDDC
MKCs
prepare blinds
Payloads
Blinds
-
blinding factors
RequestMint
Blinds
sign blinds
ResponseMint
BlindSignatures
unblind
Coins
transfer, e.g. using Signal
CoinStack



Description

This is a high level description of the actual steps, details follow in the chapters in Details.[3]

Participants

Issuer can mint, refresh and redeem coins. This entity will probably an account handling system (a.k.a. bank) behind it for doing actual real-world payments. The issuer is trusted to handle coins and payments correctly, but is *not trusted* regarding privacy - the target of OpenCoin is to protect the privacy of the transfers.

Alice and **Bob** are clients using OpenCoin. Technically they are software clients, and they represent the *users* of the system.[4] They need to be known by the customer in order to mint or redeem coins. Authentication could be required to renew coin. This would allow a "closed" system, in which accounts of the users could be monitored.

Steps in the protocol

create CDDC

The issuer creates a pair of cryptographic keys (the currency keys), and signs a *Currency Description Document Certificate* (CDDC) with its secret key. This contains information about the currency, like denominations, urls but also the public key. This is the top document which establishes the trust in all other elements.

Not mentioned in the CDDC but probably somewhere on the issuer website is the relation between opencoins and actual real-world money. Let's say the currency of an example issuer is called "opencent".[5] The rule might be that one opencent is given out for one EUR cent, and redeemed for one EUR cent, effectively binding the opencent to the EUR.

create MKCs

For each denomination in the currency separate minting keys are generated, and a *Mint Key Certificate* (MKC) for them as well. Those MKCs are signed the secret currency key. The mint keys are only valid for a defined period of time.[6]

RequestCDDCSerial

RequestCDDCSerial asks for the current serial number of the CDDC. The currency description could change over time, maybe because urls have changed. Every time a new CDDC is created, with a new, increasing serial number. The clients need to make sure to always use the most current CDDC, but they can cache it, allowing them to skip the next step.

ResponseCDDCSerial

ResponseCDDSerial contains the current serial of the CDDC.

RequestCDDC

RequestCDDC asks for a CDDC. If no serial is provided, the message asks for the most current CDDC.

ResponseCDDC

ResponseCDDC contains the CDDC

RequestMKCs

RequestMKCs asks for the Mint Key Certificates. The client can specify specific denominations or *mint key ids*. An unspecified request will return all current MKCs.

ResponseMKCs

ResponseMCKs contains the MKCs

prepare blinds

This step prepares a coin. In essence this is a Payload with a serial number, which is later on signed by the issuer using a denomination specific mint key. The "envelope" mentioned above really means that the serial is blinded using a separate random secret **blinding factor** for each serial number. This factor is needed later on to "open up the envelope", reversing the blinding operation. Hence, the client has to store the blinding factor for later on. As the blinding factor is individual for each serial number, a reference number is created to reference serial, blinding factor and Blind.

The blinds contain the reference, the blind to be signed, and the mint key id for the denomination or value of the coin.

RequestMint

RequestMint hands in the Blinds created in the step before, asking for the blind to be signed.

Most likely the issuer has authenticated the client. The mint key id tells the issuer what denomination to use for the signing operation. This will allow the issuer to deduct a payment for the minting operation (outside OpenCoin).

The message also carries a transaction_reference (a random number), in case there is a delay in the minting process. The client can then later on ask again for the signatures to be delivered using the same transaction_reference.

sign blinds

The issuer uses the secret minting key for the desired operation to sign the Blind, creating Blind Signatures.

ResponseMint

ResponseMint contains the Blind Signatures for the Blinds.

unblind

The client will unblind the Blind Signature using the before stored secret blinding factor. This gives the client the signature for the serial number, and both together give the Coin.

CoinStack

When sending coins multiple coins can be combined into a CoinStack. This CoinStack can also have a "subject", maybe containing an order reference - the reason the CoinStack is handed over in the first place.

The transfer of the CoinStack is out of scope of the OpenCoin protocol. We imagine multiple ways: using a messenger like Signal, using email or using the Browser. A CoinStack can also be encoded using a QR code, and maybe printed out and sent using normal postal mail.

Anyhow, the point of this step is that Alice transfers a CoinStack to Bob. And because she is a fair user, she will delete all coins that were contained in the CoinStack on her side.

tokenize

Coins that are received need to be swapped for new ones, in order to protect the receiver against double spending. Bob needs to decide which new coins he wants to have, and tokenize amount in the right way to have a good selection of future coin sizes. [7]

prepare blinds

Knowing the right coin selection from the step before Bob prepares Blinds the same way Alice has done with hers, creating Payloads (containing serials), and storing the blinding secrets under a reference.

RequestRenew

The renewal process is effectively the same as in minting new coins, but it is paid for in opencoins, instead of making a payment in the background using accounts. Hence, the RequestRenew message needs to contain Coins that have a value that matches the sum of value of the Blinds. The message also contains a transaction_reference in case a delay happens.

ResponseDelay

This step is optional.

If something takes a while at the issuer when signing the blinds, either while handling a RequestMint or RequestRenew a ResponseDelay can be sent back to indicate that the client should try again sometime later. This allows the network connection be closed in the meantime. Hopefully operations resume in a short time.

Delays should be avoided on the issuer side.

RequestResume

Bob will try a suitable amount of time later on to resume the transaction (the renewal in this case). The RequestResume message will send over the transaction reference, and the issuer will hopefully respond with a ResponseMint message, or with another ResponseDelay.

validate coins

Bob validates the Coins, just as Alice did.

RequestRedeem

Bob might want to swap some or all of the Coins he holds for real-world currency at the issuer. He sends in the coins in a RequestRedeem message. This effectively takes the coins out of circulation, and the issuer will make a payment to Bob's account. This requires the client to be authenticated for this step, which again is outside the OpenCoin protocol.

ResponseRedeem

The issuer confirms that everything went ok using the ResponseRedeem message.

Details

Cryptographic operations

- hashes
- signatures

Field Reference

Field Types

This lists all the fields used in the protocol. All:

- String: A JSON string.
- Integer: A JSON integer.
- BigInt: A JSON string containing a large number represented as the hex representation of it.
- DateTime: A JSON string containing the ISO representation of a date.
- List: A JSON list that can contain all the possible field types mentioned here.
- URL: A JSON string containing the URL of a resource.
- WeightedURLList: A list of 2 element tuples [Int, URL]. Useful for round-robin, but also
  reflects a preference. The lower the higher the priority.
- Schema / Object: A JSON object that conforms to the given schema.

All fields are mandatory, but can be empty in case of strings.

Fields

additional_info field

A field where the issuer can store additional information about the currency.

Type: String
Used in: CDDC

blind_signature field

The signature on the blinded hash of a payload.

Type: BigInt
Used in: ResponseMint Message

blind_signatures field

A list of BlindSignatures.

Type: List of BlindSignatures
Used in: ResponseMint Message

blinded_payload_hash field

The blinded hash of a payload.

Type: BigInt
Used in: Blind

blinds field

A List of Blinds

Type: List of Blinds
Used in: RequestMint Message, RequestRenew Message

cdd field

Contains the Currency Description Document (CDD)

Type: CDD
Used in: CDDC

cdd_expiry_date field

When does the CDD expire?

The cdd should not be used or validated after this date.

Type: String
Used in: CDDC

cdd_location field

Hint to download the CDD if not available anyway.

Useful for clients to “bootstrap” a yet unknown currency.

Type: URL
Used in: CDDC, Payload

cdd_serial field

The version of the CDD.

Should be increased by 1 on a new version.

Type: Int
Used in: CDDC, MKC, RequestCDDC Message, ResponseCDDSerial Message

cdd_signing_date field

When was the CDD signed?

Type: DateTime
Used in: CDDC

cddc field

A full Currency Description Document Certificate.

Type: CDDC
Used in: ResponseCDDC Message

coins field

A list of coins.

Type: List of Coins
Used in: CoinStack Message, RequestRedeem Message, RequestRenew Message

coins_expiry_date field

Coins expire after this date.

Do not use coins after this date. the coins before this date.

Type: DateTime
Used in: MKC

currency_divisor field

Used to express the value in units of 'currency name'.

Example: a divisor of 100 can be used express cent values for EUR or USD.

Type: Int
Used in: CDDC

currency_name field

The name of the currency, e.g. Dollar.

Use the name of the 'full' unit, and not its fraction, e.g. 'dollar' instead of 'cent', and use the currency_divisor to express possible fractions.

Type: String
Used in: CDDC

denomination field

The value of the coin(s).

Type: Int
Used in: MKC, Payload

denominations field

The list of possible denominations.

Should be chosen wisely and listed in increasing value.

Type: List of Int
Used in: CDDC, RequestMKCs Message

id field

Identifier, a somewhat redundant hash of the PublicKey

This is just a visual helper, and MUST not be relied on. Calculate the hash of the key in the client.

Type: BigInt
Used in: CDD, CDDC, MintKey, MKC

info_service field

A list of locations where more information about the currency can be found.

This refers to human-readable information.

Type: WeightedURLList
Used in: CDDC

issuer_cipher_suite field

Identifier of the cipher suite that is used.

The format is: HASH-SIGN-BLINDING, e.g. SHA512-RSA-CHAUM83

Type: String
Used in: CDDC

field: issuer_id

Id (hash) of the issuer public master key in the CDDC

Type: BigInt
Used in: MKC, Payload

issuer_public_master_key field

The hash of the issuer's public key

The only valid identifier of a currency is the master key.

Type: PublicKey
Used in: CDDC

keys field

A list of Mint Key Certificates

Type: List of MKCs
Used in: ResponseMKCs Message

message_reference field

Client internal message reference

Set by the client, echoed by the issuer.

Type: Integer
Used in: RequestCDDSerial Message, RequestCDDC Message, RequestMint Message, RequestMKCs Message, RequestRedeem Message, RequestRenew Message, RequestResume Message, ResponseCDDSerial Message, ResponseCDDC Message, ResponseDelay Message, ResponseMint Message, ResponseMKCs Message, ResponseRedeem Message

mint_key field

The mint key that was signed in the certificate

Type: MintKey
Used in: MKC

mint_key_id field

Identifier of the mint key used.

Type: BigInt
Used in: Blind, Payload

mint_key_ids field

What mint keys should be returned?

If left emtpy, no filter is applied.

Type: List of BigInt
Used in: RequestMKCs Message

mint_service field

A list of locations where Blinds can be minted into Coins

Type: WeightedURLList
Used in: CDDC

modulus field

The modulus of the public key

Type: BigInt
Used in: PublicKey

payload field

The payload of the coin.

Type: Payload
Used in: Coin

protocol_version field

The protocol version that was used.

Type: Url
Used in: CDDC, Payload

public_exponent field

The exponent of the public key.

Type: BigInt
Used in: PublicKey

public_mint_key field

The public key of the mint key.

Type: PublicKey
Used in: MintKey

redeem_service field

A list of locations where Coins can be redeemed.

Type: WeightedURLList
Used in: CDDC

reference field

An identifier that connects Blind, BlindSignature and blinding secrets.

Set by the client, echoed by the server.

Type: String
Used in: ResponseMint Message, Blind

renew_service field

A list of locations where Coins can be renewed.

Type: WeightedURLList
Used in: CDDC

serial field

The serial of the Coin.

This random value is generated by clients. It is used to identify coins and prevent double spending.
Once the coin is spent, the serial will be stored by the issuer. Because of its sufficient long length it is supposed to be unique for each coin. A high entropy (crypto grade quality) is important.

Type: BigInt
Used in: Payload

sign_coins_not_after field

Use MintKey only before this date.

Type: DateTime
Used in: MKC

sign_coins_not_before field

Use MintKey only after this date.

Type: String
Used in: MKC

signature field

A signature within a certificate.

Type: String
Used in: CDDC, Coin, MKC

status_code field

The issuer can return a status code, like in HTTP
2XX SUCCESS
3XX DELAY / TEMPORARY ERROR
4XX PERMANENT ERROR

Type: Integer
Used in: ResponseCDDSerial Message, ResponseCDDC Message, ResponseDelay Message, ResponseMint Message, ResponseMKCs Message, ResponseRedeem Message

status_description field

Description that the issuer passes along with the status_code.

Type: String
Used in: ResponseCDDSerial Message, ResponseCDDC Message, ResponseDelay Message, ResponseMint Message, ResponseMKCs Message, ResponseRedeem Message

subject field

A message that can be passed along with the coin stack.

Can be left empty. Used informally to indicate a reason for payment etc.

Type: String
Used in: CoinStack Message

transaction_reference field

A random identifier that allows the client to resume a delayed mint/renew process.

This should be a good random number.

Type: BigInt
Used in: RequestMint Message, RequestRenew Message, RequestResume Message

type field

String identifying the type of message.

This is the id that is used for parsing the message.
One of:

- blinded payload hash
- blind signature
- cdd
- cdd certificate
- coin
- coinstack
- mint key certificate
- mint key
- payload
- rsa public key
- request cddc
- request cdd serial
- request mint key certificates
- request mint
- request redeem
- request renew
- request resume
- response cddc
- response cdd serial
- response delay
- response mint key certificates
- response mint
- response redeem

Type: String
Used in: ResponseMint Message, Blind, CDDC, CDDC, Coin, CoinStack Message, MKC, MKC, Payload, RequestCDDSerial Message, RequestCDDC Message, RequestMint Message, RequestMKCs Message, RequestRedeem Message, RequestRenew Message, RequestResume Message, ResponseCDDSerial Message, ResponseCDDC Message, ResponseDelay Message, ResponseMint Message, ResponseMKCs Message, ResponseRedeem Message, PublicKey

Schemata

Elements of messages, but never used standalone

CDD

Fields

- **additional_info**: A field where the issuer can store additional information about the currency. - *String*
- **cdd_expiry_date**: When does the CDD expire? - *String*
- **cdd_location**: Hint to download the CDD if not available anyway. - *URL*
- **cdd_serial**: The version of the CDD. - *Int*
- **cdd_signing_date**: When was the CDD signed? - *DateTime*
- **currency_divisor**: Used to express the value in units of 'currency name'. - *Int*
- **currency_name**: The name of the currency, e.g. Dollar. - *String*
- **denominations**: The list of possible denominations. - *List of Int*
- **id**: Identifier, a somewhat redundant hash of the PublicKey - *BigInt*
- **info_service**: A list of locations where more information about the currency can be found. - *WeightedURLList*
- **redeem_service**: A list of locations where Coins can be redeemed. - *WeightedURLList*
- **issuer_cipher_suite**: Identifier of the cipher suite that is used. - *String*
- **issuer_public_master_key**: The hash of the issuer's public key - *PublicKey*
- **protocol_version**: The protocol version that was used. - *Url*
- **renew_service**: A list of locations where Coins can be renewed. - *WeightedURLList*
- **type**: String identifying the type of message. - *String*
- **mint_service**: A list of locations where Blinds can be minted into Coins - *WeightedURLList*

Example

```json
{
  "additional_info": "",
  "cdd_expiry_date": "2023-07-11T08:39:38.421080",
  "cdd_location": "https://opencent.org",
  "cdd_serial": 1,
  "cdd_signing_date": "2022-07-11T08:39:38.421080",
  "currency_divisor": 100,
  "currency_name": "OpenCent",
  "denominations": [1, 2, 5],
  "id": "96fb652e249ddb9045f04fe64ba9663893ed46cf2bc117eb2674cbe09312762e",
  "info_service": [
        [10, "https://opencent.org"]
      ],
  "issuer_cipher_suite": "RSA-SHA512-CHAUM86",
  "issuer_public_master_key": {
    "modulus":
"a45b9342b15deff8a5ba1a0dce50c06a0d34ac8ab251d0cf62ff6db825a714b57fcb8b243862ae539c3e997ebefc31c9983a6300ea08b81a4f613447f9123829",
    "public_exponent": 65537,
    "type": "rsa public key"
  },
  "mint_service": [
        [10, "https://opencent.org"],
    [20, "https://opencent.com/validate"]
      ],
  "protocol_version": "https://opencoin.org/1.0",
  "redeem_service": [
        [10, "https://opencent.org"]
      ],
  "renew_service": [
        [10, "https://opencent.org"]
      ],
  "type": "cdd"
}
```

Source

CDDC

Fields

- **cdd**: Contains the Currency Description Document (CDD) - *CDD*
- **signature**: A signature within a certificate. - *String*
- **type**: String identifying the type of message. - *String*

Example

```
{
  "cdd": {
    "additional_info": "",
    "cdd_expiry_date": "2023-07-11T08:39:38.421080",
    "cdd_location": "https://opencent.org",
    "cdd_serial": 1,
    "cdd_signing_date": "2022-07-11T08:39:38.421080",
    "currency_divisor": 100,
    "currency_name": "OpenCent",
    "denominations": [1, 2, 5],
    "id": "96fb652e249ddb9045f04fe64ba9663893ed46cf2bc117eb2674cbe09312762e",
    "info_service": [
        [10, "https://opencent.org"]
      ],
    "issuer_cipher_suite": "RSA-SHA512-CHAUM86",
    "issuer_public_master_key": {
      "modulus": "a45b9342b15deff8a5ba1a0dce50c06a0d34ac8ab251d0cf62ff6db825a714b57fcb8b243862ae539c3e997ebefc31c9983a6300ea08b81a4f613447f9123829",
      "public_exponent": 65537,
      "type": "rsa public key"
    },
    "mint_service": [
        [10, "https://opencent.org"],
      [20, "https://opencent.com/validate"]
      ],
    "protocol_version": "https://opencoin.org/1.0",
    "redeem_service": [
        [10, "https://opencent.org"]
      ],
    "renew_service": [
        [10, "https://opencent.org"]
      ],
    "type": "cdd"
  },
  "signature": "1e38b379d5259d7d09094a458955b3892d36aa98bd40ff6625ffd15d145da2d7f3997360ceb9d9d86a004e362249f01dd7c35779ae79987121430402f8d43c5d",
  "type": "cdd certificate"
}
```

Source

PublicKey

Fields

- **modulus**: The modulus of the public key - *BigInt*
- **public_exponent**: The exponent of the public key. - *BigInt*
- **type**: String identifying the type of message. - *String*

Example

```
{
  "modulus": "a45b9342b15deff8a5ba1a0dce50c06a0d34ac8ab251d0cf62ff6db825a714b57fcb8b243862ae539c3e997ebefc31c9983a6300ea08b81a4f613447f9123829",
  "public_exponent": 65537,
  "type": "rsa public key"
}
```

Source

See MKC

MintKey

Fields

- **cdd_serial**: The version of the CDD. - *Int*
- **coins_expiry_date**: Coins expire after this date. - *DateTime*
- **denomination**: The value of the coin(s). - *Int*
- **id**: Identifier, a somewhat redundant hash of the PublicKey - *BigInt*
- **issuer_id**: Id (hash) of the issuer public master key in the CDDC - *BigInt*
- **public_mint_key**: The public key of the mint key. - *PublicKey*
- **sign_coins_not_after**: Use MintKey only before this date. - *DateTime*
- **sign_coins_not_before**: Use MintKey only after this date. - *String*
- **type**: String identifying the type of message. - *String*

Example

```
{
  "cdd_serial": 1,
  "coins_expiry_date": "2023-10-19T08:39:38.421080",
  "denomination": 1,
  "id": "e3e053d4fa03ba6b857051bbd3b9f7c7b0c05e42f6260712139450e18b2c94bc",
  "issuer_id": "96fb652e249ddb9045f04fe64ba9663893ed46cf2bc117eb2674cbe09312762e",
  "public_mint_key": {
    "modulus": "9cf7454307e7cb5a7cee5bf46eeb0e214daeed0fd9ad35192434af31195fc32b190ec32c2270517a59856397242601b365ca57bf3b64dd66e3c61f7d253e2dbf",
    "public_exponent": 65537,
    "type": "rsa public key"
  },
  "sign_coins_not_after": "2023-07-11T08:39:38.421080",
  "sign_coins_not_before": "2022-07-11T08:39:38.421080",
  "type": "mint key"
}
```

Source

MKC

A *Mint Key Certificate*.

Fields

- **mint_key**: The mint key that was signed in the certificate - *MintKey*
- **signature**: A signature within a certificate. - *String*
- **type**: String identifying the type of message. - *String*

Example

```
{
  "mint_key": {
    "cdd_serial": 1,
    "coins_expiry_date": "2023-10-19T08:39:38.421080",
    "denomination": 1,
    "id": "e3e053d4fa03ba6b857051bbd3b9f7c7b0c05e42f6260712139450e18b2c94bc",
    "issuer_id": "96fb652e249ddb9045f04fe64ba9663893ed46cf2bc117eb2674cbe09312762e",
    "public_mint_key": {
      "modulus": "9cf7454307e7cb5a7cee5bf46eeb0e214daeed0fd9ad35192434af31195fc32b190ec32c2270517a5985639 7242601b365ca57bf3b64dd66e3c61f7d253e2dbf",
      "public_exponent": 65537,
      "type": "rsa public key"
    },
    "sign_coins_not_after": "2023-07-11T08:39:38.421080",
    "sign_coins_not_before": "2022-07-11T08:39:38.421080",
    "type": "mint key"
  },
  "signature": "4c33d555c14e0728ac0e5676fcc4e1665abb7db1f4ec9bd0173bd52dea0801ab81c63742bd82b243a4464aa 119178a9af8c23504fabc77ef9268167c2443e231",
  "type": "mint key certificate"
}
```

Source

Payload

Fields

- **cdd_location**: Hint to download the CDD if not available anyway. - *URL*
- **denomination**: The value of the coin(s). - *Int*
- **issuer_id**: Id (hash) of the issuer public master key in the CDDC - *BigInt*
- **mint_key_id**: Identifier of the mint key used. - *BigInt*
- **protocol_version**: The protocol version that was used. - *Url*
- **serial**: The serial of the Coin. - *BigInt*
- **type**: String identifying the type of message. - *String*

Example

```
{
  "cdd_location": "https://opencent.org",
  "denomination": 1,
  "issuer_id": "96fb652e249ddb9045f04fe64ba9663893ed46cf2bc117eb2674cbe09312762e",
  "mint_key_id": "e3e053d4fa03ba6b857051bbd3b9f7c7b0c05e42f6260712139450e18b2c94bc",
  "protocol_version": "https://opencoin.org/1.0",
  "serial": "947c8ab50b05b92e7949362b1e714b833ac2748ab9df43d7a3b7a9c33a5a46eb",
  "type": "payload"
}
```

Source

Blind

Fields

- **blinded_payload_hash**: The blinded hash of a payload. - *BigInt*
- **mint_key_id**: Identifier of the mint key used. - *BigInt*
- **reference**: An identifier that connects Blind, BlindSignature and blinding secrets. - *String*
- **type**: String identifying the type of message. - *String*

Example

```
{
  "blinded_payload_hash":
"9920380bd19718b319833af6166b5a22d9c1265084e4a2ab08740e2e9ccdbbabce575c14ead9d97783b0095d97023471ff018cc7d334fbdc8a13a3281a2cc3fc",
  "mint_key_id": "e3e053d4fa03ba6b857051bbd3b9f7c7b0c05e42f6260712139450e18b2c94bc",
  "reference": "a0",
  "type": "blinded payload hash"
}
```

Source

BlindSignature

Fields

- **blind_signature**: The signature on the blinded hash of a payload. - *BigInt*
- **reference**: An identifier that connects Blind, BlindSignature and blinding secrets. - *String*
- **type**: String identifying the type of message. - *String*

Example

```
{
  "blind_signature":
"470f61eefcf2fa82afd216a618df615d9a8a4d26047b448d94f61c6dd2514cb93032e17b842e63747723ed9b
c729934012597207f269be208e437709455f10f9",
  "reference": "a0",
  "type": "blind signature"
}
```

Source

Coin

Fields

- **payload**: The payload of the coin. - *Payload*
- **signature**: A signature within a certificate. - *String*
- **type**: String identifying the type of message. - *String*

Example

```
{
  "payload": {
    "cdd_location": "https://opencent.org",
    "denomination": 1,
    "issuer_id": "96fb652e249ddb9045f04fe64ba9663893ed46cf2bc117eb2674cbe09312762e",
    "mint_key_id": "e3e053d4fa03ba6b857051bbd3b9f7c7b0c05e42f6260712139450e18b2c94bc",
    "protocol_version": "https://opencoin.org/1.0",
    "serial": "947c8ab50b05b92e7949362b1e714b833ac2748ab9df43d7a3b7a9c33a5a46eb",
    "type": "payload"
  },
  "signature":
"50be266f4644cd09e7dc222b64b1e002f1760290f13ff7ddbd52fcdf3d8b3c1ef8b981e68b655f5be128526
aba718775ab947f322c074031a3380cdb5dd1feb",
  "type": "coin"
}
```

Source

RequestCDDSerial Message

Fields

- **message_reference**: Client internal message reference - *Integer*
- **type**: String identifying the type of message. - *String*

Example

```
{
  "message_reference": 100000,
  "type": "request cdd serial"
}
```

Source

ResponseCDDSerial Message

- **cdd_serial**: The version of the CDD. - *Int*
- **message_reference**: Client internal message reference - *Integer*
- **status_code**: The issuer can return a status code, like in HTTP - *Integer*
- **status_description**: Description that the issuer passes along with the status_code. - *String*
- **type**: String identifying the type of message. - *String*

Fields

Example

```
{
  "cdd_serial": 1,
  "message_reference": 100000,
  "status_code": 200,
  "status_description": "ok",
  "type": "response cdd serial"
}
```

Source

RequestCDDC Message

- **cdd_serial**: The version of the CDD. - *Int*
- **message_reference**: Client internal message reference - *Integer*
- **type**: String identifying the type of message. - *String*

Fields

Example

```
{
  "cdd_serial": 1,
  "message_reference": 100001,
  "type": "request cddc"
}
```

Source

ResponseCDDC Message

Fields

- **cddc**: A full Currency Description Document Certificate. - *CDDC*
- **message_reference**: Client internal message reference - *Integer*
- **status_code**: The issuer can return a status code, like in HTTP - *Integer*
- **status_description**: Description that the issuer passes along with the status_code. - *String*
- **type**: String identifying the type of message. - *String*

Example

```
{
  "cddc": {
    "cdd": {
      "additional_info": "",
      "cdd_expiry_date": "2023-07-11T08:39:38.421080",
      "cdd_location": "https://opencent.org",
      "cdd_serial": 1,
      "cdd_signing_date": "2022-07-11T08:39:38.421080",
      "currency_divisor": 100,
      "currency_name": "OpenCent",
      "denominations": [1, 2, 5],
      "id": "96fb652e249ddb9045f04fe64ba9663893ed46cf2bc117eb2674cbe09312762e",
      "info_service": [
        [10, "https://opencent.org"]
      ],
      "issuer_cipher_suite": "RSA-SHA512-CHAUM86",
      "issuer_public_master_key": {
        "modulus": "a45b9342b15deff8a5ba1a0dce50c06a0d34ac8ab251d0cf62ff6db825a714b57fcb8b243862ae539c3e997ebefc31c9983a6300ea08b81a4f613447f9123829",
        "public_exponent": 65537,
        "type": "rsa public key"
      },
      "mint_service": [
        [10, "https://opencent.org"],
        [20, "https://opencent.com/validate"]
      ],
      "protocol_version": "https://opencoin.org/1.0",
      "redeem_service": [
        [10, "https://opencent.org"]
      ],
      "renew_service": [
        [10, "https://opencent.org"]
      ],
      "type": "cdd"
    },
    "signature": "1e38b379d5259d7d09094a458955b3892d36aa98bd40ff6625ffd15d145da2d7f3997360ceb9d9d86a004e362249f01dd7c35779ae79987121430402f8d43c5d",
    "type": "cdd certificate"
  },
  "message_reference": 100001,
  "status_code": 200,
  "status_description": "ok",
  "type": "response cddc"
}
```

Source

RequestMKCs Message

- **denominations**: The list of possible denominations. - *List of Int*
- **message_reference**: Client internal message reference - *Integer*
- **mint_key_ids**: What mint keys should be returned? - *List of BigInt*
- **type**: String identifying the type of message. - *String*

Fields

Example

```
{
  "denominations": [1, 2, 5],
  "message_reference": 100002,
  "mint_key_ids": [],
  "type": "request mint key certificates"
}
```

Source

ResponseMKCs Message

- **keys**: A list of Mint Key Certificates - *List of MKCs*
- **message_reference**: Client internal message reference - *Integer*
- **status_code**: The issuer can return a status code, like in HTTP - *Integer*
- **status_description**: Description that the issuer passes along with the status_code. - *String*
- **type**: String identifying the type of message. - *String*

Fields

Example

```
{
  "keys": [
    {
      "mint_key": {
        "cdd_serial": 1,
        "coins_expiry_date": "2023-10-19T08:39:38.421080",
        "denomination": 1,
        "id": "e3e053d4fa03ba6b857051bbd3b9f7c7b0c05e42f6260712139450e18b2c94bc",
        "issuer_id": "96fb652e249ddb9045f04fe64ba9663893ed46cf2bc117eb2674cbe09312762e",
        "public_mint_key": {
          "modulus":
"9cf7454307e7cb5a7cee5bf46eeb0e214daeed0fd9ad35192434af31195fc32b190ec32c2270517a5985639
7242601b365ca57bf3b64dd66e3c61f7d253e2dbf",
          "public_exponent": 65537,
          "type": "rsa public key"
        },
        "sign_coins_not_after": "2023-07-11T08:39:38.421080",
        "sign_coins_not_before": "2022-07-11T08:39:38.421080",
        "type": "mint key"
      },
      "signature": "4c33d555c14e0728ac0e5676fcc4e1665abb7db1f4ec9bd0173bd52dea0801ab81c63742bd82b243a4464aa119178a9af8c23504fabc77ef9268167c2443e231",
      "type": "mint key certificate"
    },
    {
      "mint_key": {
        "cdd_serial": 1,
        "coins_expiry_date": "2023-10-19T08:39:38.421080",
        "denomination": 2,
        "id": "3f3a34c8df9abaee2030bb7aee86bfdb896f85affc9a01fec984deb06c077c62",
        "issuer_id": "96fb652e249ddb9045f04fe64ba9663893ed46cf2bc117eb2674cbe09312762e",
        "public_mint_key": {
          "modulus": "9d218bbeaf2a8a2074e208082f8f02f91b8afd20909e023a347a97a3c0b56059013148ae9f81c3aa242c3822682e572dab24a74bd344a651ef7f06fc1493bc91",
          "public_exponent": 65537,
          "type": "rsa public key"
        },
        "sign_coins_not_after": "2023-07-11T08:39:38.421080",
        "sign_coins_not_before": "2022-07-11T08:39:38.421080",
        "type": "mint key"
      },
      "signature": "1ad4952f625078188de83c4f76ef7a47a696056a4f7acefe24f2309867775fcce3fbbf87f76944a1a8a7bc4277c532720c5fd13c4cbf65eb23d222401b4b99c9",
      "type": "mint key certificate"
    },
    {
      "mint_key": {
        "cdd_serial": 1,
        "coins_expiry_date": "2023-10-19T08:39:38.421080",
        "denomination": 5,
        "id": "2aaa99a9ffbd0377b46d757bbf82d4f65b4f05a170a2da23743066849c403776",
        "issuer_id": "96fb652e249ddb9045f04fe64ba9663893ed46cf2bc117eb2674cbe09312762e",
        "public_mint_key": {
          "modulus": "acd3ad0b8e8bf5c27f9b9c00d3057e0be8617837f1a33427dd3eaffe00f9f785d5632268d919ac5623d55f173b47eb975cb103dad8886b771f13f0b804405f81",
          "public_exponent": 65537,
          "type": "rsa public key"
        },
        "sign_coins_not_after": "2023-07-11T08:39:38.421080",
        "sign_coins_not_before": "2022-07-11T08:39:38.421080",
        "type": "mint key"
      },
      "signature": "153660ad076ded345b99701cf62ca50473ec7f80e76193cf4a5cf476175e9ca3d366859590d1dbd56e31447bb319b98f5c368d181478e011c135eacb66b99b03",
      "type": "mint key certificate"
    }
  ],
  "message_reference": 100002,
  "status_code": 200,
  "status_description": "ok",
  "type": "response mint key certificates"
}
```

Source

RequestMint Message

Fields

- **blinds**: A List of Blinds - *List of Blinds*
- **message_reference**: Client internal message reference - *Integer*
- **transaction_reference**: A random identifier that allows the client to resume a delayed mint/renew process. - *BigInt*
- **type**: String identifying the type of message. - *String*

Example

```
{
  "blinds": [
    {
      "blinded_payload_hash":
"9920380bd19718b319833af6166b5a22d9c1265084e4a2ab08740e2e9ccdbbabce575c14ead9d97783b0095
d97023471ff018cc7d334fbdc8a13a3281a2cc3fc",
      "mint_key_id": "e3e053d4fa03ba6b857051bbd3b9f7c7b0c05e42f6260712139450e18b2c94bc",
      "reference": "a0",
      "type": "blinded payload hash"
    },
    {
      "blinded_payload_hash":
"20e5236334720449c4bd930900e607804711b6cff41fefbeea8ed7f79a6093a29b27a74541bbc6b1967984f
88347c24f34b4e2463ca0250deb78f3f982b21a70",
      "mint_key_id": "3f3a34c8df9abaee2030bb7aee86bfdb896f85affc9a01fec984deb06c077c62",
      "reference": "a1",
      "type": "blinded payload hash"
    },
    {
      "blinded_payload_hash": "46a495a7a62536885c244c4fb79eaa1ae0b6d9ba2a66157a0558727502abd6a57861a387f52a3e040799dd8dcc758083b00cfc312ace4fc4eebbc787bf742448",
      "mint_key_id": "2aaa99a9ffbd0377b46d757bbf82d4f65b4f05a170a2da23743066849c403776",
      "reference": "a2",
      "type": "blinded payload hash"
    }
  ],
  "message_reference": 100003,
  "transaction_reference": "f2b96ff94fe2becca5ff385cb6989de4f6f06eefdc64a909906e56d11a5460ad",
  "type": "request mint"
}
```

Source

ResponseMint Message

Fields

- **blind_signatures**: A list of BlindSignatures. - *List of BlindSignatures*
- **message_reference**: Client internal message reference - *Integer*
- **status_code**: The issuer can return a status code, like in HTTP - *Integer*
- **status_description**: Description that the issuer passes along with the status_code. - *String*
- **type**: String identifying the type of message. - *String*

Example

```
{
  "blind_signatures": [
    {
      "blind_signature": "470f61eefcf2fa82afd216a618df615d9a8a4d26047b448d94f61c6dd2514cb93032e17b842e63747723ed9bc729934012597207f269be208e437709455f10f9",
      "reference": "a0",
      "type": "blind signature"
    },
    {
      "blind_signature": "4e04c0ab289c65364e382ddb3b6a5ba4b4efc7ab2761d6f32c8bc3802c26d9704a1e6a4191a788ecbc4985f1e9a45ddc904447733d50a4f65597cba85cd2252b",
      "reference": "a1",
      "type": "blind signature"
    },
    {
      "blind_signature": "8cebe20e9464e4afe22d8db136a6a7dcf75ed868a1d3683b4a888bdd0e98cfa84a0edc00963317667850b9ce06df8048f617a58bda8bd340c935ba309e8ce5bd",
      "reference": "a2",
      "type": "blind signature"
    }
  ],
  "message_reference": 100003,
  "status_code": 200,
  "status_description": "ok",
  "type": "response mint"
}
```

Source

CoinStack Message

Fields

- **coins**: A list of coins. - *List of Coins*
- **subject**: A message that can be passed along with the coin stack. - *String*
- **type**: String identifying the type of message. - *String*

Example

```
{
  "coins": [
    {
      "payload": {
        "cdd_location": "https://opencent.org",
        "denomination": 1,
        "issuer_id": "96fb652e249ddb9045f04fe64ba9663893ed46cf2bc117eb2674cbe09312762e",
        "mint_key_id": "e3e053d4fa03ba6b857051bbd3b9f7c7b0c05e42f6260712139450e18b2c94bc",
        "protocol_version": "https://opencoin.org/1.0",
        "serial": "947c8ab50b05b92e7949362b1e714b833ac2748ab9df43d7a3b7a9c33a5a46eb",
        "type": "payload"
      },
      "signature": "50be266f4644cd09e7dc222b64b1e002f1760290f13ff7ddbd52fcdf3d8b3c1ef8b981e68b655f5be128526aba718775ab947f322c074031a3380cdb5dd1feb",
      "type": "coin"
    },
    {
      "payload": {
        "cdd_location": "https://opencent.org",
        "denomination": 2,
        "issuer_id": "96fb652e249ddb9045f04fe64ba9663893ed46cf2bc117eb2674cbe09312762e",
        "mint_key_id": "3f3a34c8df9abaee2030bb7aee86bfdb896f85affc9a01fec984deb06c077c62",
        "protocol_version": "https://opencoin.org/1.0",
        "serial": "e2c1c0093af6535f3d801e811d904b2891ef35a65d0e87955397db3db993ca59",
        "type": "payload"
      },
      "signature": "66c2befe37d30744691fec39a9056c530f439a4879c9a388018409ece20446b342bc7889f40efad4674d778b9f130b13a39bea03a214c80c4703bf0ba4993294",
      "type": "coin"
    },
    {
      "payload": {
        "cdd_location": "https://opencent.org",
        "denomination": 5,
        "issuer_id": "96fb652e249ddb9045f04fe64ba9663893ed46cf2bc117eb2674cbe09312762e",
        "mint_key_id": "2aaa99a9ffbd0377b46d757bbf82d4f65b4f05a170a2da23743066849c403776",
        "protocol_version": "https://opencoin.org/1.0",
        "serial": "c58356123496667ac18786f09123be1bf9116596ad30ceb676c34c8c439f1377",
        "type": "payload"
      },
      "signature": "6ae0dcc4af9e47e4e4478b7f2ec172645fa744fa7dcf55a4544ab01eadd601188099e6c27dc441231b73913a95f33e64915f5a4d43c1e21f5fd6f27a61a5304e",
      "type": "coin"
    }
  ],
  "subject": "a little gift",
  "type": "coinstack"
}
```

Source

RequestRenew Message

Coins that are received need to be swapped for new ones, in order to protect the receiver against double spending. Otherwise, the sender could keep a copy of the coins and try to use the coins again. Before doing so we need to ask: what coin sizes should be chosen for the coins to be minted?

What if we have not the right coin selection for an amount to pay? Imagine that the price is 5 opencent, but we just have coins in the sizes: 2, 2, 2.

One solution would be to require the recipient to give change. This would make the protocol more complicated, and would just shift the problem to the recipient. Another approach is to allow partial spending coins, but this again makes the protocol more complicated.[8]

The easy way out is to aim for a selection of coins that allows us to pay *any* amount below or equal to the sum of all coins. E.g. if we own the value of 6 opencent it would be advisable to have coin selection 2,2,1,1 in order to pay all possible amounts. This also prevents *amount tracing*, where an awkward price (13.37) asks for an awkward coin exchange at the issuer beforehand.

So, we need to look at the combined sum of coins received and coins already in possession, and needs to find the right coin selection to be able to make all possible future coin transfers. We will then know which coins to keep, and what blinds to make and paying for the minting using *all* the just received coins and using *some* existing coins.

Fields

- **blinds**: A List of Blinds - *List of Blinds*
- **coins**: A list of coins. - *List of Coins*
- **message_reference**: Client internal message reference - *Integer*
- **transaction_reference**: A random identifier that allows the client to resume a delayed mint/renew process. - *BigInt*
- **type**: String identifying the type of message. - *String*

Example

```
{
  "blinds": [
    {
      "blinded_payload_hash": "2186fc8b344ad4339214d8d3784fccd75e39d6f39e9c85457f95c663a091147fe5afb03097566ca976bd2ca0a7c0690679cdfdd33906112773b46f6d8a7aa4db",
      "mint_key_id": "3f3a34c8df9abaee2030bb7aee86bfdb896f85affc9a01fec984deb06c077c62",
      "reference": "b0",
      "type": "blinded payload hash"
    },
    {
      "blinded_payload_hash": "468410cb8c4b386a89c2bc2daa263132c704c99eeb4bfa26e37310800dd809f7659b9d1c38ec624858cf52324adc121cab3444344657f117e2e49ee274c229ff",
      "mint_key_id": "3f3a34c8df9abaee2030bb7aee86bfdb896f85affc9a01fec984deb06c077c62",
      "reference": "b1",
      "type": "blinded payload hash"
```

```json
    },
    {
      "blinded_payload_hash": "ab7d98df621495a7d4c190bf1aecc6693a2ae7c3d5ded687fa8643f815f7029a46f8cc1581cc73d1566e9345a5559ee7c52b25c44c438ede424d17660b3abf3",
      "mint_key_id": "3f3a34c8df9abaee2030bb7aee86bfdb896f85affc9a01fec984deb06c077c62",
      "reference": "b2",
      "type": "blinded payload hash"
    },
    {
      "blinded_payload_hash": "9917a14a00dc164c65aad8ad9041c15d6525c7b2d6ce5e1374b67392f354ab8b1f633482dc9da6570b874313b501c5e7cf0051654e7d619d6008539f2f8704cf",
      "mint_key_id": "3f3a34c8df9abaee2030bb7aee86bfdb896f85affc9a01fec984deb06c077c62",
      "reference": "b3",
      "type": "blinded payload hash"
    }
  ],
  "coins": [
    {
      "payload": {
        "cdd_location": "https://opencent.org",
        "denomination": 1,
        "issuer_id": "96fb652e249ddb9045f04fe64ba9663893ed46cf2bc117eb2674cbe09312762e",
        "mint_key_id": "e3e053d4fa03ba6b857051bbd3b9f7c7b0c05e42f6260712139450e18b2c94bc",
        "protocol_version": "https://opencoin.org/1.0",
        "serial": "947c8ab50b05b92e7949362b1e714b833ac2748ab9df43d7a3b7a9c33a5a46eb",
        "type": "payload"
      },
      "signature": "50be266f4644cd09e7dc222b64b1e002f1760290f13ff7ddbd52fcdf3d8b3c1ef8b981e68b655f5be128526aba718775ab947f322c074031a3380cdb5dd1feb",
      "type": "coin"
    },
    {
      "payload": {
        "cdd_location": "https://opencent.org",
        "denomination": 2,
        "issuer_id": "96fb652e249ddb9045f04fe64ba9663893ed46cf2bc117eb2674cbe09312762e",
        "mint_key_id": "3f3a34c8df9abaee2030bb7aee86bfdb896f85affc9a01fec984deb06c077c62",
        "protocol_version": "https://opencoin.org/1.0",
        "serial": "e2c1c0093af6535f3d801e811d904b2891ef35a65d0e87955397db3db993ca59",
        "type": "payload"
      },
      "signature": "66c2befe37d30744691fec39a9056c530f439a4879c9a388018409ece20446b342bc7889f40efad4674d778b9f130b13a39bea03a214c80c4703bf0ba4993294",
      "type": "coin"
    },
    {
      "payload": {
        "cdd_location": "https://opencent.org",
        "denomination": 5,
        "issuer_id": "96fb652e249ddb9045f04fe64ba9663893ed46cf2bc117eb2674cbe09312762e",
        "mint_key_id": "2aaa99a9ffbd0377b46d757bbf82d4f65b4f05a170a2da23743066849c403776",
        "protocol_version": "https://opencoin.org/1.0",
        "serial": "c58356123496667ac18786f09123be1bf9116596ad30ceb676c34c8c439f1377",
        "type": "payload"
      },
      "signature": "6ae0dcc4af9e47e4e4478b7f2ec172645fa744fa7dcf55a4544ab01eadd601188099e6c27dc441231b73913a95f33e64915f5a4d43c1e21f5fd6f27a61a5304e",
      "type": "coin"
    }
  ],
  "message_reference": 100004,
  "transaction_reference": "a88b8ba5519ed859d7bfa076d4c12937",
  "type": "request renew"
}
```

Source

ResponseDelay Message

Fields

- **message_reference**: Client internal message reference - *Integer*
- **status_code**: The issuer can return a status code, like in HTTP - *Integer*
- **status_description**: Description that the issuer passes along with the status_code. - *String*
- **type**: String identifying the type of message. - *String*

Example

```
{
  "message_reference": 100004,
  "status_code": 300,
  "status_description": "ok",
  "type": "response delay"
}
```

Source

RequestResume Message

Fields

- **message_reference**: Client internal message reference - *Integer*
- **transaction_reference**: A random identifier that allows the client to resume a delayed mint/renew process. - *BigInt*
- **type**: String identifying the type of message. - *String*

Example

```
{
  "message_reference": 100005,
  "transaction_reference": "a88b8ba5519ed859d7bfa076d4c12937",
  "type": "request resume"
}
```

Source

RequestRedeem Message

Fields

- **coins**: A list of coins. - *List of Coins*
- **message_reference**: Client internal message reference - *Integer*
- **type**: String identifying the type of message. - *String*

Example

```
{
  "coins": [
    {
      "payload": {
        "cdd_location": "https://opencent.org",
        "denomination": 2,
        "issuer_id": "96fb652e249ddb9045f04fe64ba9663893ed46cf2bc117eb2674cbe09312762e",
        "mint_key_id":
"3f3a34c8df9abaee2030bb7aee86bfdb896f85affc9a01fec984deb06c077c62",
        "protocol_version": "https://opencoin.org/1.0",
        "serial": "f8411ca510e367d831937739342158cbf09db8d34817f0dabddb80622ac784d7",
```

```json
        "type": "payload"
      },
      "signature": "5d4c7ca9e97426aeb59f2fe22220e164f1b37f0931920494a359fdbc30d90064c843a04e0d16dda3dc1d83814b8bac934873a0c5317fc04b51cf3a912967d8b8",
      "type": "coin"
    },
    {
      "payload": {
        "cdd_location": "https://opencent.org",
        "denomination": 2,
        "issuer_id": "96fb652e249ddb9045f04fe64ba9663893ed46cf2bc117eb2674cbe09312762e",
        "mint_key_id": "3f3a34c8df9abaee2030bb7aee86bfdb896f85affc9a01fec984deb06c077c62",
        "protocol_version": "https://opencoin.org/1.0",
        "serial": "cca2647b1fd803e5fe5f0131489d450a1b5980ffa739da18b3b143ca8bd6fc79",
        "type": "payload"
      },
      "signature": "5bb381233bd3a0c9437ce7009b81263784fddf594a9843f611a752355025fdaf04368b51be50f311de9fb288b6ae927b50aa2885f64ca54bfba629d08c19c5c4",
      "type": "coin"
    }
  ],
  "message_reference": 100006,
  "type": "request redeem"
}
```

Source

ResponseRedeem Message

Fields

- **message_reference**: Client internal message reference - *Integer*
- **status_code**: The issuer can return a status code, like in HTTP - *Integer*
- **status_description**: Description that the issuer passes along with the status_code. - *String*
- **type**: String identifying the type of message. - *String*

Example

```
{
  "message_reference": 100006,
  "status_code": 200,
  "status_description": "ok",
  "type": "response redeem"
}
```

Source

Appendix

FAQ

Scope

Having said all of the above, we scope the protocol, and it's description in the following way:

Targeted at developers - developers should be enabled (and motivated) by the OpenCoin protocol to implement standard confirming software components and apps. However, we hope that this documentation is also understandable for the interested user (or founder, investor, auditor, etc.)

Just the protocol - we don't deliver any ready to use implementations. This allows us to fully focus on the protocol, and keeps a separation to actual implementations.

Easy to understand - we try to avoid complexity. This affects the protocol itself as well as its documentation. This means: if you, the reader, don't understand a sentence or a concept, please contact us. We will improve the description. Being easy to understand is one of the main goals of OpenCoin.

Only the core - lots of developments have happened since we started. Take the example of messengers like Signal, Telegram or WhatsApp. The have opened new ways to transport messages, and they take care of identifying the communication partner. This especially means that message transport and authentication stays out of scope.

History and old results

- Project history
- Project papers
- Crypto report
- Legal report
- Code bases (v1, sandbox, javascript implementation)

Artifacts

Details of the documents in the artifacts directory

JSON Schemata

Ideas

- use .oc file ending
- oc over html
- opencoin.org as a web interface demo provider, that can handle .oc files

Building blocks for writing

create CDDC
create MKCs
RequestCDDSerial
ResponseCDDSerial
RequestCDDC
ResponseCDDC
RequestMKCs
ResponseMKCs
prepare blinds
RequestMint
sign blinds
ResponseMint
unblind
transfer CoinStack, e.g. using Signal
tokenize sum
prepare blinds
RequestRenew
ResponseDelay
RequestResume
validate coins
RequestRedeem
ResponseRedeem

Header

Fields

Example

```
```

Source

Request

Fields

Example

```
```

Source

Response

Fields

Example

```
```

Source

1. David Chaum, “Blind signatures for untraceable payments”, Advances in Cryptology - Crypto ‘82, Springer-Verlag (1983), 199-203. ↩

2. Please check with your lawyer if this is a good idea. ↩

3. It is easier to follow along with the above diagram open in a second window (or printout). ↩

4. To keep the diagram simple we have left out Charlene who was mentioned above in "How does it work?". Bob does everything she does. ↩

5. "opencent" refers to the specific example currency. The generic term "opencoin" refers to any currency following the OpenCoin protocol (of which opencent is one). ↩

6. This is to minimize damage in case the mint keys get compromised. ↩

7. It might be that also some existing coins might be needed to be swapped to get a good coin selection. See Renew. ↩

8. GNU Taler experiments with this approach: in essence coins don't have serials but keys, which can sign a partial amount to be spent. This requires more smartness to avoid double spending, introducing new problems to be solved. ↩