OpenCoin

a protocol for privacy preserving electronic cash payments

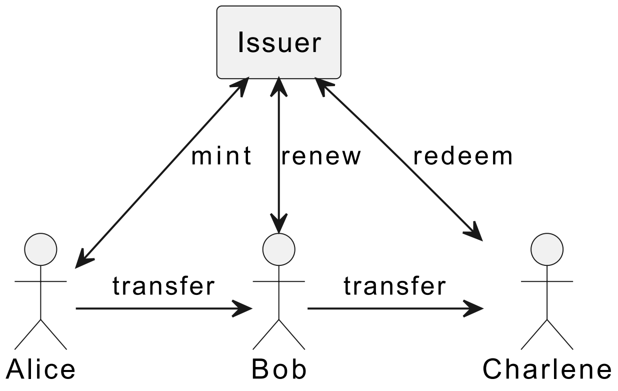
Table of contents

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. taxability.

The trade-off seems to be that their system is harder is more complex and harder to understand. We also doubt that this complexities are necessary to reach the stated goals. We also doubt that the goals can really be reached, and also find that the systems 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.

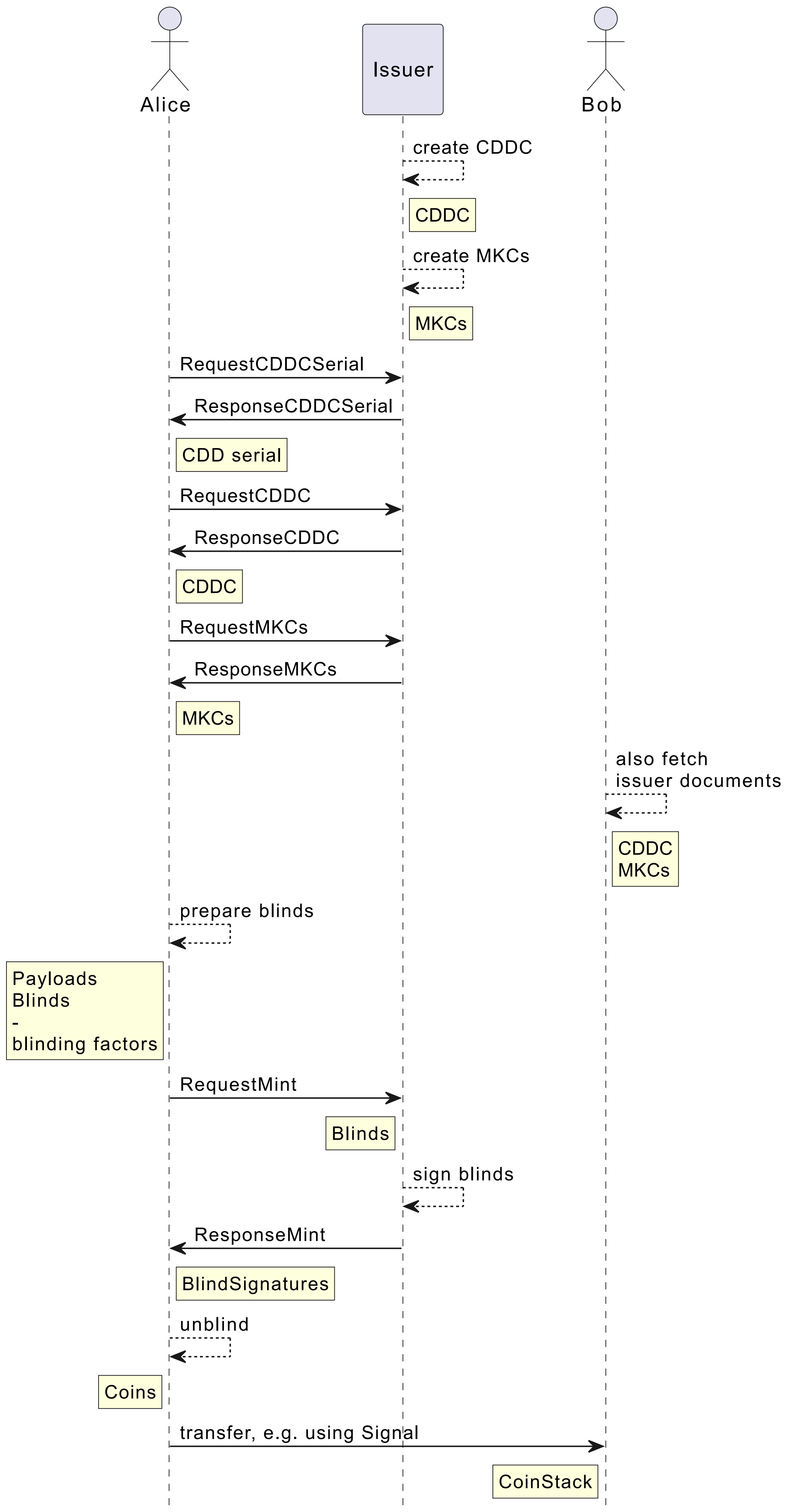
The OpenCoin protocol

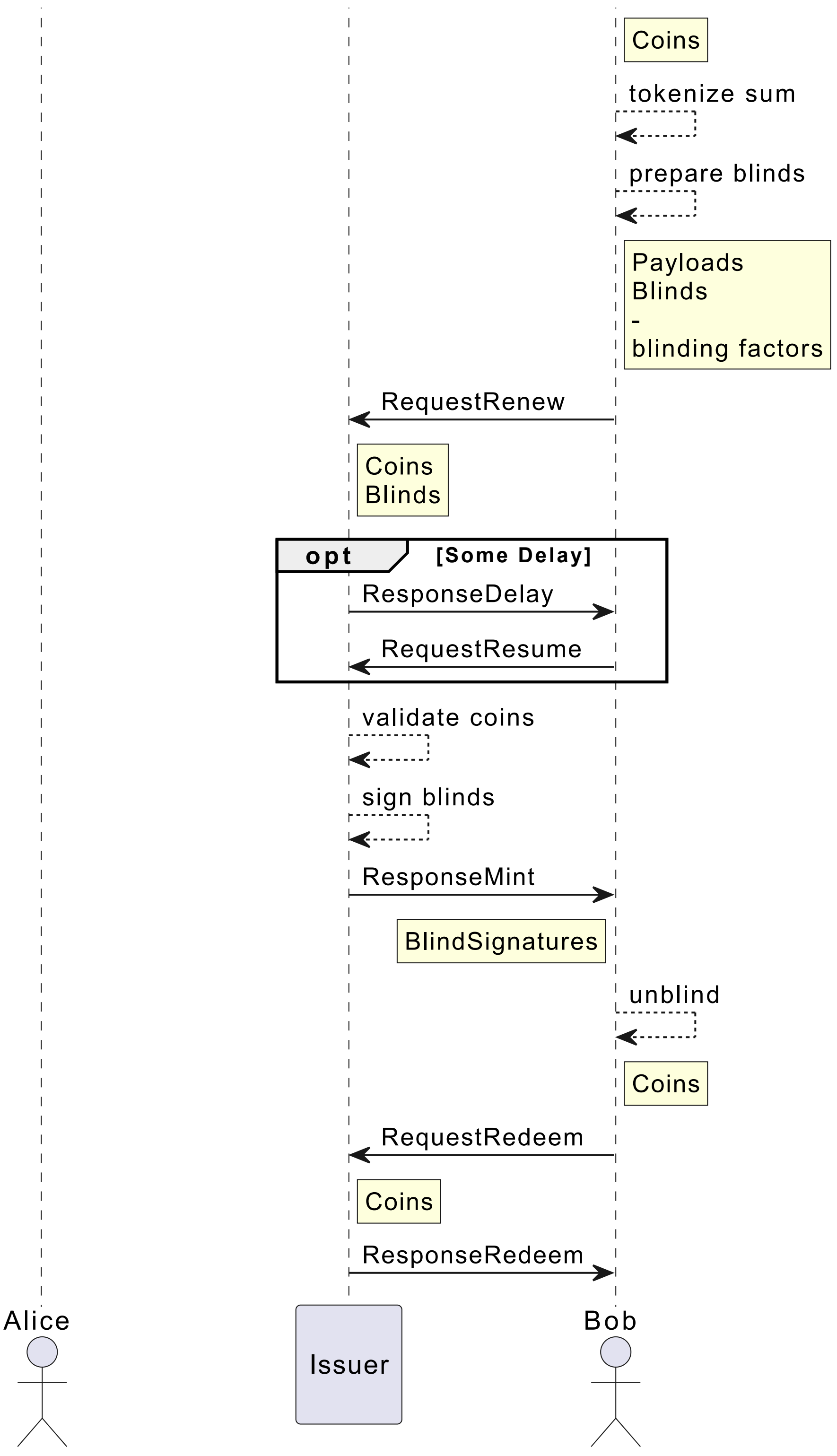
Assumptions

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.

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* in regards of 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.[3] 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

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". 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.[4]

RequestCDDCSerial

This message 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

This message contains the **CDDC serial**.

RequestCDDC

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

ResponseCDDC

This message contains the **CDDC**

RequestMKCs

With this message the client 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

This reply 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** contains the reference, the blind to be signed, and the mint key id for the denomination or value of the coin.

RequestMint

This message 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 the **blind signatures**.

ResponseMint

This message contains the **blind signatures** for the blinds.

unblind

The client will unblind the 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 sum

prepare blinds

RequestRenew

ResponseDelay

RequestResume

validate coins

RequestRedeem

ResponseRedeem

Problems

- Tax
- Money laundering
- Blackmail and other crime

Details

Cryptographic operations

- hashes
- signatures

Building blocks

Elements of messages, but never used standalone

CDD

```
{
  "additional_info": "",
  "cdd_expiry_date": "2023-07-08T15:32:04.103469",
  "cdd_location": "https://opencent.org",
  "cdd_serial": 1,
  "cdd_signing_date": "2022-07-08T15:32:04.103469",
  "currency_divisor": 100,
  "currency_name": "OpenCent",
  "denominations": [1, 2, 5],
  "id": "b7520e474a9b80032c051cca120dc52048d3f334fc5b0d9e12c9aca155e90d3a",
  "info_service": [
        [10, "https://opencent.org"]
      ],
  "invalidation_service": [
        [10, "https://opencent.org"]
      ],
  "issuer_cipher_suite": "RSA-SHA512-CHAUM86",
  "issuer_public_master_key": {
    "modulus": "9f4e001f979e03a9c755694f936bc68930a67813e1d8b3afb4d9de408088522d551eec8babcc2ef99fc1f2814d49aa0c3e497f05d77fcd932192c742caf0adaf",
    "public_exponent": 65537,
    "type": "rsa public key"
  },
  "protocol_version": "https://opencoin.org/1.0",
  "renewal_service": [
        [10, "https://opencent.org"]
      ],
  "type": "cdd",
  "validation_service": [
        [10, "https://opencent.org"],
    [20, "https://opencent.com/validate"]
  ]
}
```

Source

Messages

CDDSerial

CDDC

```
{
  "cdd": {
    "additional_info": "",
    "cdd_expiry_date": "2023-07-08T15:32:04.103469",
    "cdd_location": "https://opencent.org",
    "cdd_serial": 1,
    "cdd_signing_date": "2022-07-08T15:32:04.103469",
    "currency_divisor": 100,
    "currency_name": "OpenCent",
    "denominations": [1, 2, 5],
    "id":
"b7520e474a9b80032c051cca120dc52048d3f334fc5b0d9e12c9aca155e90d3a",
    "info_service": [
        [10, "https://opencent.org"]
      ],
    "invalidation_service": [
        [10, "https://opencent.org"]
      ],
    "issuer_cipher_suite": "RSA-SHA512-CHAUM86",
    "issuer_public_master_key": {
      "modulus": "9f4e001f979e03a9c755694f936bc68930a67813e1d8b3afb4d9de408088522d551eec8babcc2ef99fc1f2814d49aa0c3e497f05d77fcd932192c742caf0adaf",
      "public_exponent": 65537,
      "type": "rsa public key"
    },
    "protocol_version": "https://opencoin.org/1.0",
    "renewal_service": [
        [10, "https://opencent.org"]
      ],
    "type": "cdd",
    "validation_service": [
        [10, "https://opencent.org"],
      [20, "https://opencent.com/validate"]
    ]
  },
  "signature": "51287b6abca69924f8ab9ad121fb3a70fad0b4133d4c72e6e7142eb11ebdb770ac855b90301b124343ed97365dad40e62f0784fb7194f15e01b436d97d917318",
  "type": "cdd certificate"
}
```

Source

MKCs

Mint

CoinStack

Renew

Resume

Redeem

Reference

Appendix

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 it's 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

bla [2]

Footnotes

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

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