# Security report milestone 1

Group 1, Class p2
Davide Cruz 71776
João Aniceto 72255

## Assumptions

- Each server has a non-certified, asymmetric key pair (Die-Hellman, RSA, etc) with a well-known public component.
    - So a pair of keys is available to use right off the bat

## Blockchain

Ordered linked list of blocks where each block contains a cryptography hash of the previous blockchain

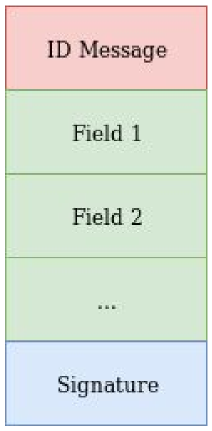
## Client Functionality

The client app first validates the CC card of the user. Then it shows the menu with the following functionality:

- Create an auction
    - Takes the auction id or show the auctions in list form
- Terminate an auction created by him
    - Takes the auction id or show the auctions in list form
- List auctions
    - Can select open ones or closed
    - Can check their status and bids if applicable when selected
- Check bids made
    - Check outcome or auction
- Make a bid
    - Select auction and input value
    - It saves the receipt on a file give by the user
- Validate receipt
    - Select file

## Message protocol

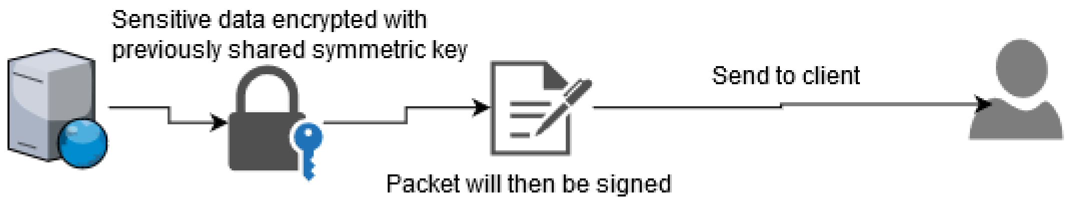
### Message structure



- **ID** identifies the what type and **purpose** the message has
- Next we have **fields** that change according to the type/id of the message
- Some messages may have an Id Sender given that we use for knowing the sender, giving that for testing we use localhost in the sockets to communicate, which makes the sender unrecognizable by just looking at the address.

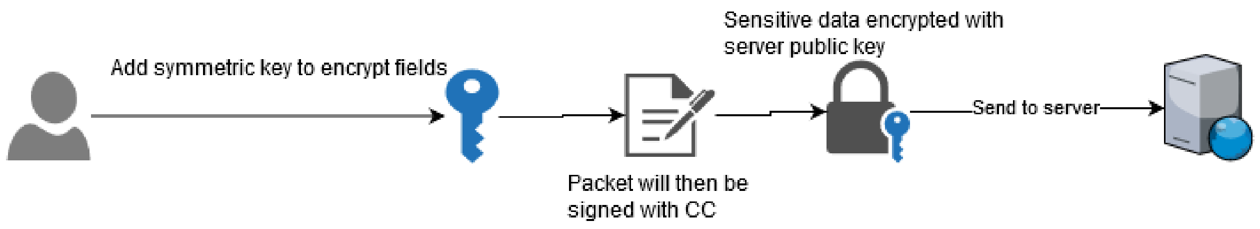
### Connection properties

UDP will be used for all communication.
The communication for servers-clients:

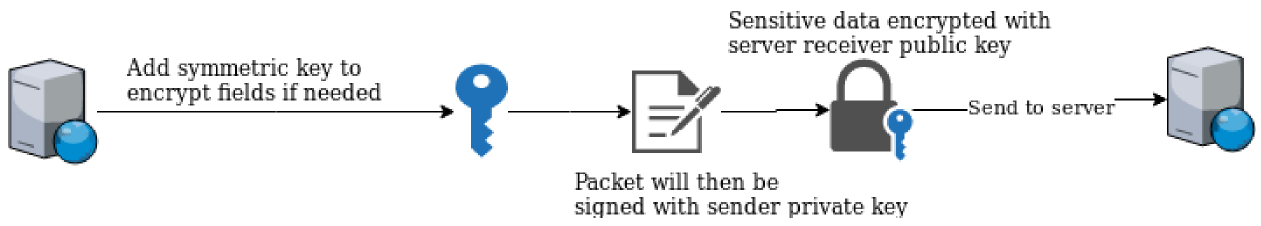

Sensitive data encrypted with previously shared symmetric key
Send to client
Packet will then be signed

Fields that are sensitive will be encrypted with a symmetric key sent in the request. Packet will be signed to assure the source and integrity.

The communication for clients-servers:


Add symmetric key to encrypt fields
Sensitive data encrypted with server public key
Send to server
Packet will then be signed with CC

The communication for manager <-> repository:


Add symmetric key to encrypt fields if needed
Sensitive data encrypted with server receiver public key
Packet will then be signed with sender private key
Send to server

## Create auction

CLIENT -> MANAGER

- Packet type:
    - ID: 1
- Packet structure:
    - signature
    - Request/Response Data
- Signature:
    - Request signed by client with CC
    - Reponse signed by manager
- Request data:
    - Auction Type
    - Claim time
    - Time limit
    - Name
    - Description
- Response data:
    - Status
        - Success
            - Packet request
        - Error
            - ID
            - Reason

MANAGER -> REPOSITORY

- Packet type:
    - ID: 2
- Packet structure:
    - signature
    - Request/Response Data
- Signature:
    - Request signed by client with CC
    - Reponse signed by manager
- Request data:

- Auction Type
- Claim time
- Time limit
- Name
- Description

Response data:

- Status
    - Success
        - Packet Request
    - Error
        - ID
        - Reason

## Terminate auction

CLIENT -> MANAGER

Packet type:

- ID: 3

Packet structure:

- Signature
- Request/Response Data

Signature:

- Request  signed by client with CC
- Reponse signed by manager

Request data:

- Auction unique ID

Response data:

- Status
    - Success
        - Packet request
    - Error
        - ID
        - Reason

MANAGER -> REPOSITORY

Packet type:

- ID: 4

Packet structure:

- signature
- Request/Response Data

Signature:

- Request signed by manager
- Reponse signed by repository

Request data:
- Auction unique ID

Response data:
- Status
    - Success
        - Packet request
    - Error
        - ID
        - Reason

## List auctions (closed/open)

Client -> Repository:

Packet type:
- ID: 5

Packet structure:
- Signature (only response)
- Request/Response Data

Signature:
- Response signed by Repository

Request data:
- Type (all, closed, open)

Response data:
- Status
    - Success
        - Packet request
    - Error
        - Reason
- Auctions
    - ID of auction
    - Type
    - Name
    - Description
    - Status (open/closed)
    - Time left
    - Claim time

## List bids

Client -> Repository:

Packet type:
- ID: 6

Packet structure:

- Signature (only response)
- Request/Response Data

Signature:
- Response signed by Repository

Request data:
- Auction ID

Response data:
- Status
    - Success
        - Packet request
    - Error
        - Reason
- Bid

## List bids of Client

Client -> Manager:

Packet type:
- ID: 7

Packet structure:
- Signature encrypted with manager public key
- Request/Response Data

Signature:
- Request signed by client
- Response signed by Repository

Request data:
- Symmetric key ***(Encrypted with manager public key)***
- Algorithm
- Mode

Response data:
- Status
    - Success
        - Packet request
    - Error
        - Reason
- Bids ***(Encrypted with symmetric key)***

Repository -> Manager:

Packet type:
- ID: 8

Packet structure:
- Request/Response Data

Signature:

- Request signed by Repository
- Response signed by Manager

Request data:

-

Response data:

- Status
    - Success
        - Packet request
    - Error
        - Reason
- Blockchains of all auctions

## Bid on auction

1º Step Crypto-Puzzle Phase

Client -> Repository:

Packet type:

- ID: 9

Packet structure:

- Signature (only response)
- Request/Response Data

Signature:

- Response signed by Repository

Request data:

Response data:

- CryptoPuzzle challenge
- Status
    - Success
        - Packet request
    - Error
        - Reason

2º Step - Bid Phase

Client -> Repository

Packet type:

- ID: 10

Packet structure:

- Signature
- Request/Response Data

Signature:

- Request signed by client but with certificate encrypted with manager public key (hybrid)
- Response signed by Repository

## English auction

Request data:

- Symmetric key ***(Encrypted)***
- Algorithm
- Mode
- Auction ID
- Value
- CryptoPuzzle result

Response data:

- Status ***(Encrypted with symmetric key)***
  - Success
  - Error
    - Reason
- Receipt ***(Encrypted with symmetric key)***

## Blind auction

Request data:

- Symmetric key ***(Encrypted)***
- Algorithm
- Mode
- Auction ID
- Value ***(Encrypted with a symmetric key generated for this bid named keyBlindBid)***
- CryptoPuzzle result

Response data:

- Status ***(Encrypted with symmetric key)***
  - Success
  - Error
    - Reason
- Receipt ***(Encrypted with symmetric key)***

3º Step - Validate bid

Repository -> Manager:

Packet type:

- ID: 11

Packet structure:

- Signature
- Request/Response Data

Signature:

- Request signed by client
- Response signed by manager

Request data:

- Bid from user signed by repository ***(Encrypted)***

Response data:

- Status ***(Encrypted with symmetric key)***
    - Success
    - Error
        - Reason
- Bid signed by manager

## Check auction outcome

Client -> Manager

### English auction

Packet type:

- ID: 12

Packet structure:

- Signature of client
- Request/Response Data

Signature:

- Request signed by client
- Response signed by

Request data:

- Symmetric key ***(Encrypted)***
- Algorithm
- Mode
- Auction ID ***(Encrypted)***

Response data:

- Status
    - Success
    - Error
        - Reason
- Winner Identity ***(Encrypted with symmetric key)*** (if revealed or if winner)
- Winner Value ***(Encrypted with symmetric key)***

### Blind auction

1st Packet

Packet structure:

- ID: 13

Request data:

- Symmetric key ***(Encrypted)***
- Algorithm
- Mode

- Auction ID

Response data:

- Status
    - Success
    - Error
        - Reason
- Status of auction ***(Encrypted with symmetric key)***

**2nd Packet** if auction is unclaimed mode

Packet structure:

- ID: 14

Request data:

- Auction ID
- keyBlindBid ***(Encrypted)***

Response data:

- Status
    - Success
        - Request packet
    - Error
        - Reason

**2nd Packet** if auction has ended

Packet structure:

- ID: 15

Request data:

- Symmetric key ***(Encrypted)***
- Algorithm
- Mode
- Auction ID

Response data:

- Status
    - Success
    - Error
        - Reason
- Winner ID ***(Encrypted with symmetric key)*** (if revealed or if winner)
- Winner Value ***(Encrypted with symmetric key)***

## Validate receipt

Client uses the internal function to validate the receipt.

## Auction states

- Open
- Claiming Mode (with timeOut defined by who created the auction)
- Closed

## CryptoPuzzles

Use of previously known crypto puzzles algorithms. In our case we'll use **hashcash** [1] which is currently most known by its use in bitcoin.

There are currently some libraries that implements this algorithm available for use in Python:

- Official implementation: http://www.hashcash.org/libs/python/
- https://gist.github.com/i3visio/388ef5154052ed8173df4b7b9eda541b
- https://github.com/antbob/hashcash-python/blob/master/hashcash.py

## Receipt

### Structure

Bid data from packet signed by Repository -> signed by Manager to validate -> signed by Repository again

### Validation

Simply verify the chain of signatures. If all the signatures are valid we have to proof that the manage and repository both validated and stored the bid.

[1] http://www.hashcash.org/