Universidade de Aveiro

Segurança

Blockchain-based auction management
First Milestone

Inês Lopes , Nº Mec:72725
João Pedro Fonseca, Nº Mec:73779

November 15 2018

Contents

1 Introduction

The main objective of this project is to implement a secure auction management system, using the principles of a Blockchain. This system, enables users to create and participate in auctions.

The blockchain-based auction management system we purpose, needs to follow 4 rules:

- Bid confidentiality, integrity and authentication
- Bid acceptance control and confirmation
- Bid author identity and anonymity
- Honesty assurance

The rules stated before need to be complemented with security mechanisms that will allow:

- Confidentiality of all transactions;
- Privacy of the clients;
- Assurance of the identity validity of each agent (clients and servers);
- Secure transmission of all information between all agents;
- secure storage of all transactions without compromising private data or information.

This report presents and explains all the security measures we need to implement, to make the auction management system safe and confidential for all participants.

2 Architecture

The auction management system has three main entities, which interact with each other: the Client and the two Servers (management and repository).

2.1 Users

The users are all the people interested in using this system. All users need to connect to a **Client** interface of the system in order to use it. Upon authenticating themselves (using Cartão de Cidadão), they will be free to use the system at will. Each user has a set of actions available:

- Create auctions
- Create bids for auctions he has permission to bid
- See the result of the auctions he has previously bid in

Each user will need to authenticate himself before the server by solving a simple challenge sent by each of the servers.

2.2 Client

The Client represents the interface where the users will connect to. They need to use it, in order to get authenticated by the servers and exchange messages with them. This means that on the connection to the Client, the user will need to provide credentials that further on will be used for authentication in the system. The public key of Cartão de Cidadão and the user's ID will be sent to the servers and stored by them.

2.3 Servers

The central part of the project. There is a Manager server, whose main goal is to validate bids and create auctions and a Repository server, which holds all information about the auctions in a Blockchain format.

- The Client may send messages for both of these servers;
- The bids are sent to the Repository, which sends a proof-of-work to the Client;
- The Repository server sends bids to the Manager for validation;
- The Client sends validations in the form of dynamic code to the Manager.

3 Security Measures

3.1 Objects/receipts confidentiality and integrity

Some of the contents in an auction creation or bid message, should be encrypted and signed, and only the client who sent the message should be able to decrypt those contents. I the contents of a bid are modified, the signature is no longer valid. Also, the user receives a receipt after placing a bid. This receipt may later be used for checking if the contents of the bid stored in the server repository are the same as the ones stated on the user's receipt.

A certificate is a secure object, independently of the channel it flows through. The content's of the message that must be hidden should work like a certificate.

If some contents are to be encrypted, then a hybrid cipher must be used. However, the symmetric key used to encrypt the cryptogram will not be sent to the server, because the client is the only one who can access the encrypted contents.

3.2 Client-Server trustfulness

When an Client communicates with a Server for the first time, the client will know know if he/she is, in fact, talking with the server, if a proper message is received.

3.3 User-Server authentication

The User must be authenticated by the server before being able to send messages and access receipts. This means that the user must provide credentials to the server during the sign-up to be able to be authenticated from that moment forward. This means also that the user should be provided the possibility of changing his Authentication credentials whenever he considers. These credentials are the ones found on the Cartão de Cidadão.

3.4 Citizen Card authentication

The best way for a user to certify to the server that he/she is himself/herself is by using a public key certificate. As the Cartão de Cidadão provides one for each user, it can be used for the authentication/signing methods.

A certificate for the authentication key is, then, obtained, and the smartcard is also used for signing of receipts (with the private authentication key).

3.5 User accessibility

The channel in which information flows might be unsecure, but there is the need to make sure that a user's enrypted message contents and receipts are only accessible by him/her. Then, there is the need to avoid the stated information to be public knowledge.

3.6 Auction flow

The Repository server must not accept bids which have a lower amount value than the previously stored bid (the previous block on the blockchain).

4 Implementation

There are two auction types to be implemented. The **english auction**, in which the identity of the bidder is hidden and the **blind auction**, in which the bid amounts are hidden. The type of auction is specified by the auction creator upon the creation of the auction. At the end of each of these auctions, the Auction Manager will decrypt the identities (english auction) or the bid amounts (blind auction). This is the only way to declare a winner.

No server knows how to decrypt an hidden content, at least until the end of an auction.

4.1 Security mechanisms implementation

- First of all, the communication between the clients and servers, and between the servers, will be processed using UDP. Everytime the server or the client receives a message, an acknowledge must follow it. The communication is synchronous.

  Each message is in the JSON format, and the beginning of the message contains the size of it (?)

- There are 3 general schemes for public key exchange:
  - Public announcement;
  - Publicly available directory;
  - Public-key certification authorities (CAs)

  In our project, we will make a public announcement of the public key.

- To exchange the messages in a secure way, we will use an hybrid cipher (simultaneous use of symmetric cipher and asymmetric cipher). The algorithm to be used by the asymmetric cipher will be RSA, and the algorithm for the symmetric cipher will be AES.

- The client will be identified in the system by one's "id". This will be obtained from the digest of the user's public key certificate, extracted from one's Citizen Card. So, the "id" is going to be used to identify the user in a unique way.

- As stated before, not all message's content must be encrypted (only the amount and identity, in a bid message). That means that all other messages and contents should be sent in plaintext. Both servers might not know who made a certain bid.

- There must be a control of integrity on the messages exchanged (non corrupted data).

  We will use an Encrypt-then-MAC architecture, in which the MAC is computed from the cryptogram.

  After the hash function (HMAC with SHA256), the digest (result) will be encrypted by the user's private key (the one from the Citizen Card) and so we can guaranty the authentication and integrity control of all messages - signature. In other words, using this MAC we can confirm that the message received by a client came from the stated sender (authenticity) and has not been changed in the process.

- When the bid is placed and the server must sent a receipt to the client, that same receipt is signed by the server using its own private key. The client must then decrypt the signature and compare the result with the hash of the received message. If they are equal, the signature is valid and the receipt was sent by the server-

  Later, the client might want to confront the data stored on the repository with the received receipt.

  In addition to the bid's content, a receipt must also store the order number (index) of the bid, which will be useful on the search of said bid inside the blockchain.

4.1.1 Public Key exchange

On the first connection between the server and the client (which can be a server), the information exchanged is sent in plaintext between both entities. This happens because no participant knows the other. In this case the first thing to do is getting information on the other participant. Normally to do this, the agents will need to exchange their public keys.

On this connection we don't expect that someone will try to act as a server, and provide their public keys as an attacker would do.

4.1.2 Storing symmetric and private keys

PKCS11 tokens ?

4.1.3 State of the servers

The repository server must store their current state in permanent storage, because there might be a failure or multiple failures.

The repository server will store each blockchain and its bids in a text file, in a text format.

4.1.4 Receipts

The receipts will also be stored in non-volatile memory, signed by the repository server. The identity and/or amount will still be encrypted inside the receipt, so that no other user might access those contents.

4.2 Message Security

As stated previously, the messages are sent in a insecure channel, and the contents that must be hidden inside of each message must themselves be secure objects.

Only at the end of the auction, those encrypted contents must be publicly exposed, so to determine the winner of an auction.

4.2.1 Message encryption

The encryption of the exchanged messages, will be processed the same way for the client-server communications and the server-server communications (manager server and repository server). Next, we will list the flow for processing the encryption. In this flow (see Figure 1), the generic **client**, who wants to send a message to the **server**, can also be a server (since there are two of them which have a different purpose).

1. The Client obtains the public key of the Server on the first connection;
2. The message is encrypted with a symmetric key (AES);
3. Generate a shared key (symmetric key), which will be later used in the HMAC function (as a HMAC key);
4. The cryptogram is processed in a hash function (HMAC SHA256);
5. Encrypt the result of the hash (digest) with the private key of Client.

The sent message will look like as follows:

$$\begin{aligned} message = cryptogram + encrypted\_sharedkey + \\ encrypt\_PK(hmacsha256(hashed\_key, (cryptogram + encrypted\_sharedkey))) \end{aligned} \quad (1)$$

The message(1) will be sent in a JSON format, where each field is well defined in order to make the decryption process easier.

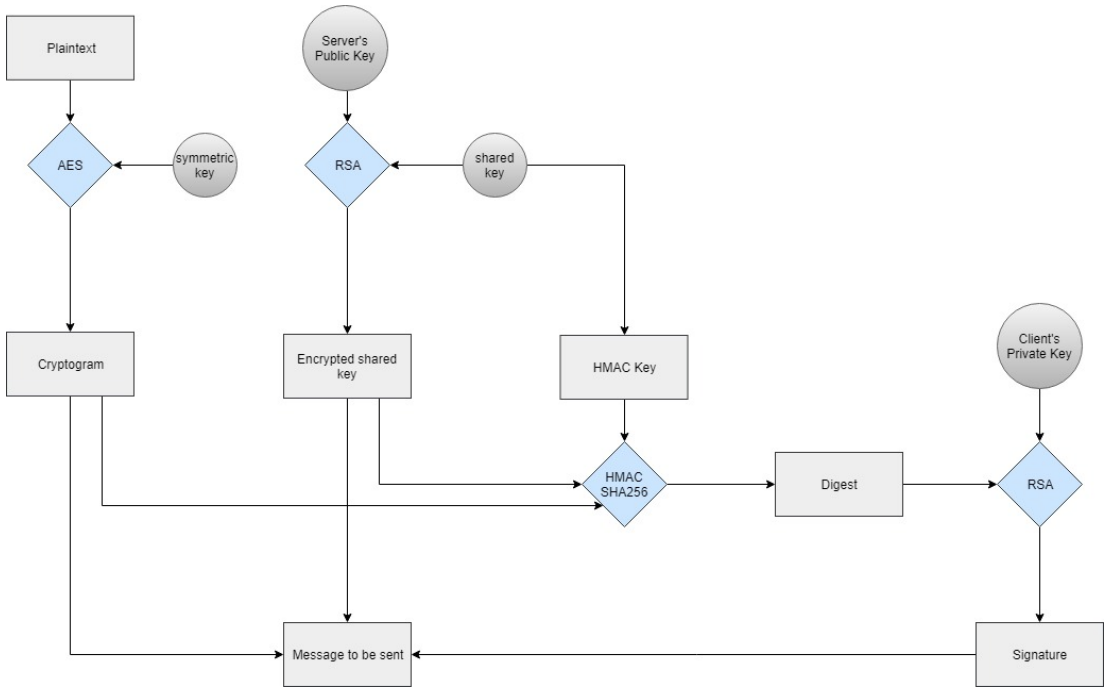

Figure 1: Content's encryption diagram.

Note: The "shared key" is a symmetric key too, but that term was used to avoid confusion with the symmetric key used to encrypt the contents.

Upon the end of the auction, the Manager server must know how to decrypt the contents of the messages sent before. The client will then send to the server the symmetric key used to create the cryptogram, using the scheme below.

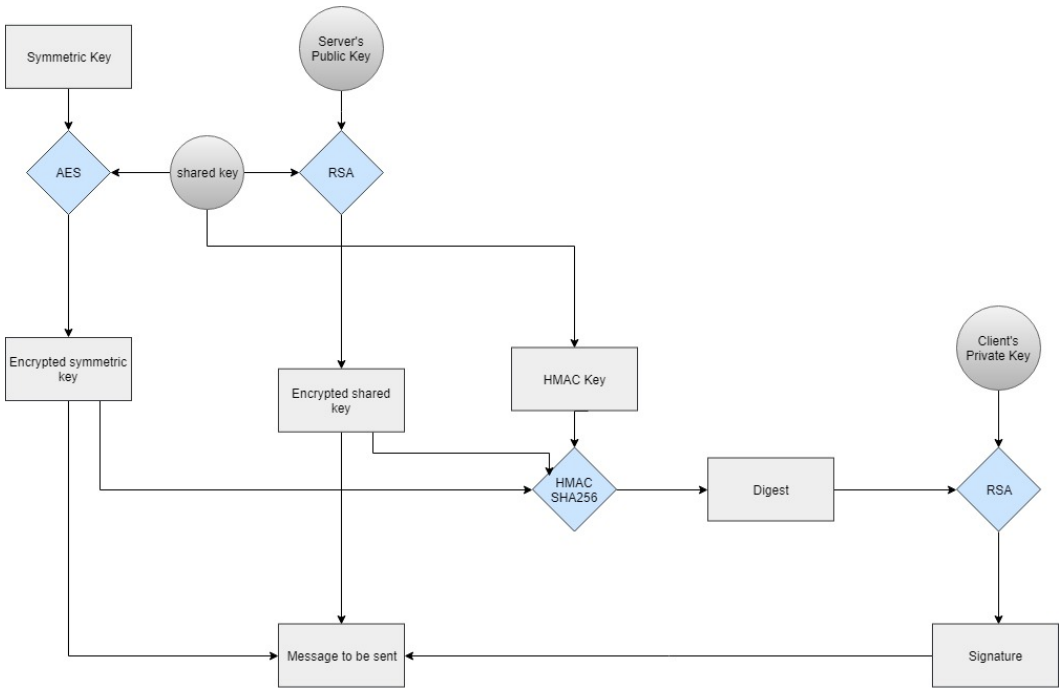

Figure 2: Symmetric key encryption diagram.

4.2.2 Message decryption

Nor the repository server nor the manager server should be able to decrypt the encrypted contents of a bid. That's why the symmetric key used to encrypt the contents is not sent inside a message (initially).

However, the manager server must publicly expose the amount and the identity of the participants at the end of an auction. To obtain the symmetric key to decrypt the previously hidden contents of the bids, the **Server** will need to follow a decryption flow:

1. Obtain Client's public key (might be already stored);
2. Decrypt the received HMAC with said key;
3. Use own private key to decrypt the symmetric key (shared key) received;
4. Compute the HMAC (SHA256) with the received encrypted symmetric key and encrypted shared key, using the decrypted shared key (.4) as the HMAC function key;
5. Compare both HMAC's (.5 with .2's result) and,
6. If the HMAC's match (authenticity confirmed), use the symmetric key to decrypt the message (data) received.

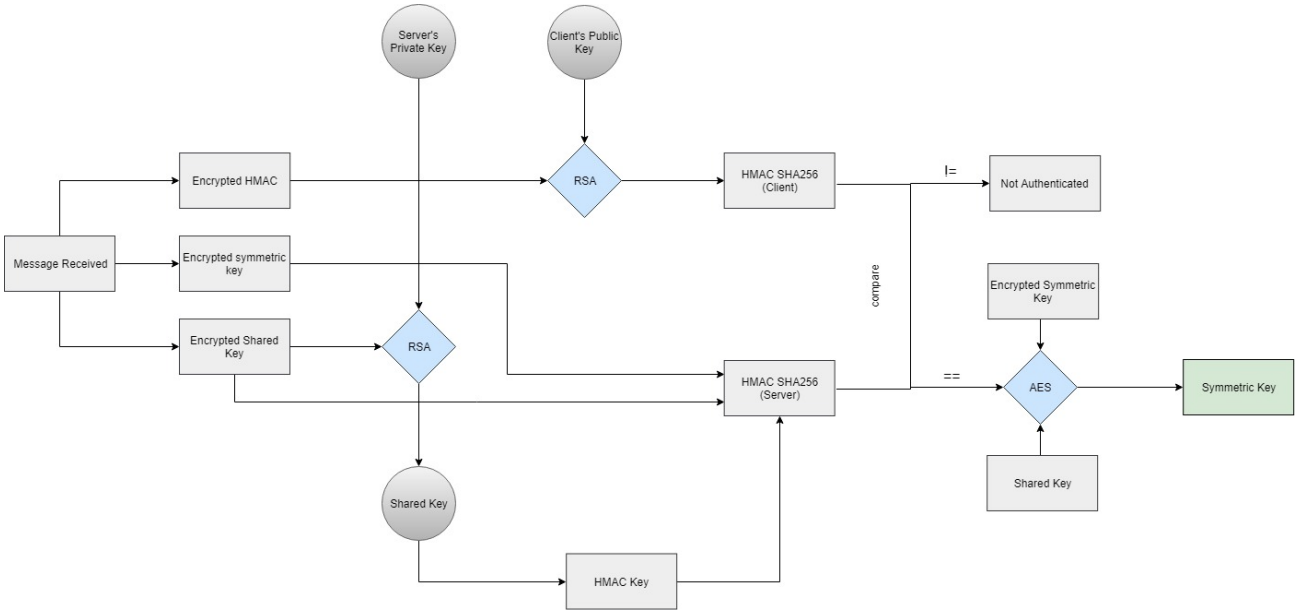

Figure 3: Message decryption diagram.

At this point, the server has all tools necessary to decrypt the hidden contents and expose them to the public.

4.3 Validation entity

Each auction will have an associated validation process. This validation occurs in the Validator entity. This validator processes the dynamic code sent by the auction creator. The result of the validation process is communicated with the auction manager.

Three main validations will be implemented:

- Limit the bidders to a given set of identities.
  - Example: {"allowed": "a,b,c..."}

- Restrict the number of bids performed by each identity
  - Example: {"restrict": {"a" : "3" , "b" : "7"}}
- State a minimum value for each bid amount value

The validator entity has access to the encrypted identities and amount values, but the Manager server doesn't. This means that the client must send the symmetric key needed for decrypting its identity and amount to the validator. The validator will also store the number of bids placed by each identity, for a current auction.

Note: The auction conditions and type must be consulted by the bidders before making a bid.

4.4 Cryptopuzzle

When a client wants to place a bid, the repository server will ask him to solve a cryptopuzzle. In this cryptopuzzle, the client will have to calculate a hash for its bid, and that hash value must be less than a certain threshold. This means that, for the bid to be accepted by the repository, its hash must be below a certain threshold given by the repository, when the client is attempting to place a bid.

Bibliography

[1] A. Zúquete. *Segurança em redes informáticas*. FCA - Editora de Informática, 2013. ISBN 978-972-722-767-9.