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 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 Message/receipts confidentiality and integrity

The content of the user's messages should be passed through a check-sum algorithm, to check the message integrity. The user's messages will also have to be confidential. This means that only the sender and the receiver must know what information they carry inside. The user's receipts must also be encrypted to hide its contents.

Each client/server will have a pair of asymmetric keys used do encrypt messages (sent to each other) through hybrid ciphers, meaning that the message will have three parts (see Section 4.2.1) This guarantees that messages between users are private and cannot be read by either the client, server or other eavesdropping entities

3.2 Client-Server trustfulness

When an Client communicates with a Server for the first time, this will send a message with a challenge, which will certify that the sender was the Server and not another entity, after being answered.

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. The authentication takes place with the use of certificates and challenges between the client and server.

3.4 Client-Server security

The communication between the Server and the Client should be "secure". This means that a message sent by the client should not be eavesdropped by someone trying to intercept the messages (Man-in-the-Middle). A Diffie-Hellman exchange will happen to create a shared secret between the two entities.

3.5 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 methods.

A certificate for the authentication key is, then, obtained, and the card is also used for signing of receipts and messages with the user's private authentication key.

3.6 User-Server security

Trusting that the channel where the information flows is secure, there is also the need to make sure that a User only accesses the messages sent or received by himself, mainly receipts, thus avoiding the information inside them to be public knowledge. This can be achieved by encrypting the messages so that only the receiver and the sender can read them.

4 Implementation

The exchange of messages between server-client and server-server must be done in a secure way. This means that all messages must be confidential, thus they can only be read by the entities which are supposed to read them.

4.1 Security mechanisms implementation

- First of all, the Client and the Server must exchange a shared key using the **Diffie-Hellman** method. This shared key will be used to encrypt the plaintext to be sent, in a symmetric key encryption scheme.
- Next, the Client and the Server must exchange their signed public key certificates (see Figure 3)
- 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.
- The 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.

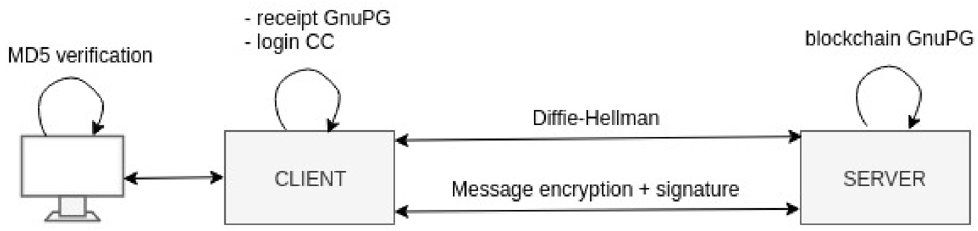

Figure 1: Client-Server interaction diagram.

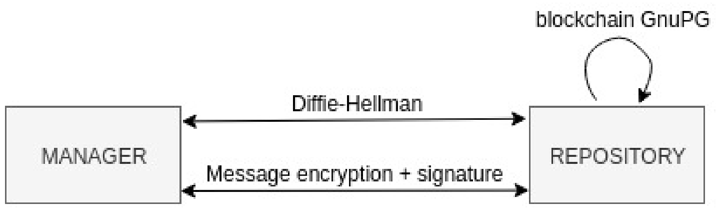

Figure 2: Server-Server interaction diagram.

4.1.1 Certificate Validation and 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 certificates (or public keys generated from the certificates).

The exchange of the certificates between the two servers and between the client and the server is processed as follows (considering Server A as a Client and Server B as a Server) (see Figure 3):

1. First, the servers must have a private key, from which they will generate a certificate. This certificate will generate a valid public key from the server which generated it.
2. The Server (might be the repository or the manager) sends its own self-signed certificate to the other Server or Client.
3. The Client will then validate the certificate chain. If it is invalid, the certificate is discarded. If it is valid, the Client will send a signed challenge encrypted with the Server's public key and its own certificate attached.
4. The Server will also validate the Client's certificate chain and if it is valid, it will reply to the challenge with a signed answer to it, which must be send encrypted with the Client's public key.

On the first connection from all parties the connection is insecure, so as the fist connection is started, we don't expect that someone will try to act as a server, and provide their certificates as a normal attacker would do.

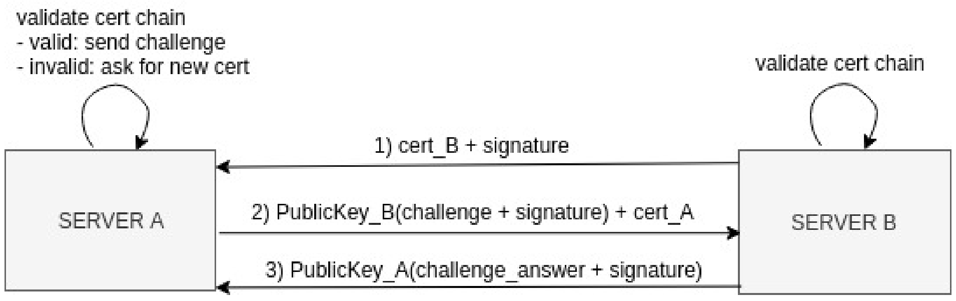
validate cert chain
- valid: send challenge
- invalid: ask for new cert
validate cert chain
1) cert_B + signature
SERVER A
2) PublicKey_B(challenge + signature) + cert_A
SERVER B
3) PublicKey_A(challenge_answer + signature)

Figure 3: Public Key certificates exchange diagram.

Upon exchanging certificates, the servers can get the public key of the other participant. This means that from now on, the communication can be done using the public keys stored in the known certificates.

4.2 Message Security

Upon knowing that the messages should be exchanged without compromising the information, we are forced to treat the channels where those message are being sent as insecure. The simplest way to share the messages on a insecure channel is to send so that only each entity can read them. We will now explain the way our system will encrypt and decrypt the messages.

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 4), 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 Diffie-Hellman symmetric key (AES) calculated for the current session;

3. Process the symmetric key with a hash function (SHA256), so it can be later used in the HMAC function;

4. The Client encrypts the original symmetric key using Server's public key (RSA);

5. The encrypted symmetric key is joined (see .5) with the encrypted message (see .3) and processed in a hash function (HMAC SHA256, using the key of .4);

6. Encrypt the result of the hash (digest) with the private key of Client.

The sent message will look like as follows:

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

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

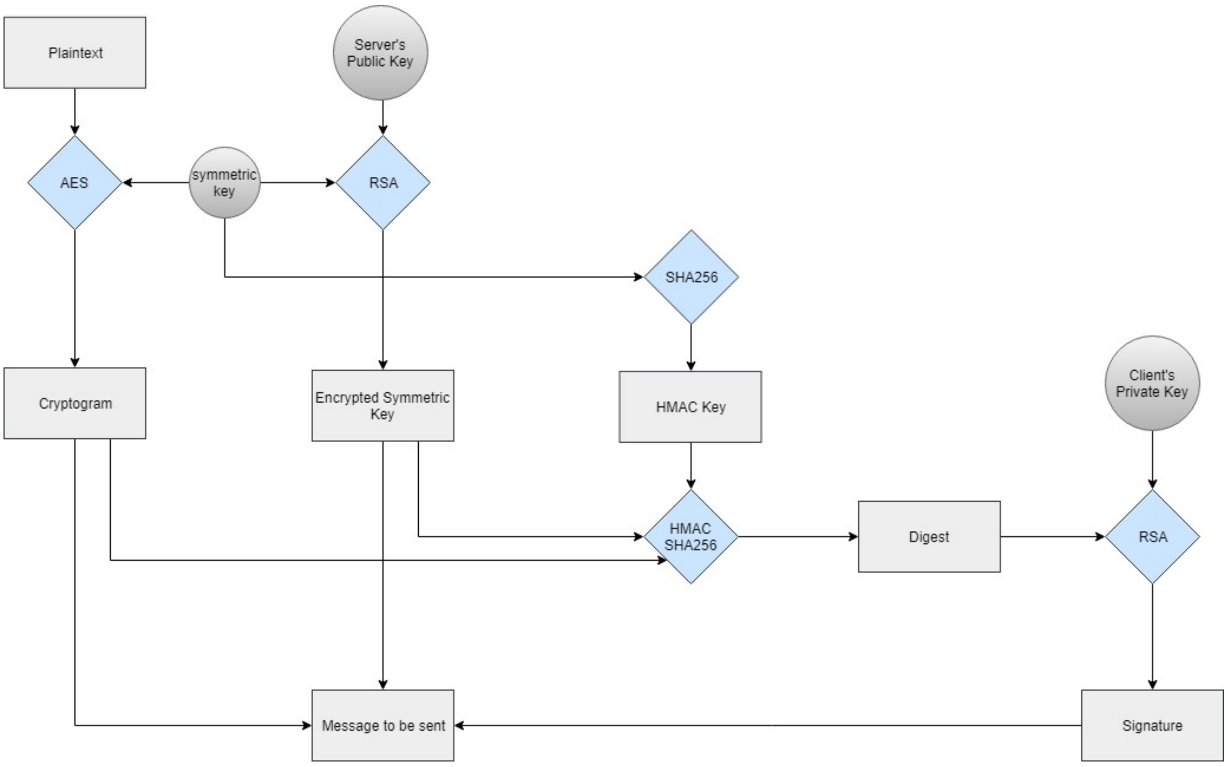

Figure 4: Message encryption diagram.

4.2.2 Message decryption

To revert the **encryption** made before, and read the contents the message, the **Server** will need to follow a decryption flow:

1. Obtain Client's public key;
2. Decrypt the received HMAC with said key;
3. Use his private key to decrypt the symmetric key received;
4. Compute the Hash (SHA256) of the symmetric key;
5. Compute the HMAC (SHA256) with the received encrypted message and encrypted symmetric key, using the result of the hash of the decrypted symetric key (.4) as the HMAC function key;
6. Compare both HMAC's (.5 with .2's result) and,
7. If the HMAC's match (authenticity confirmed), use the symmetric key to decrypt the message (data) received.

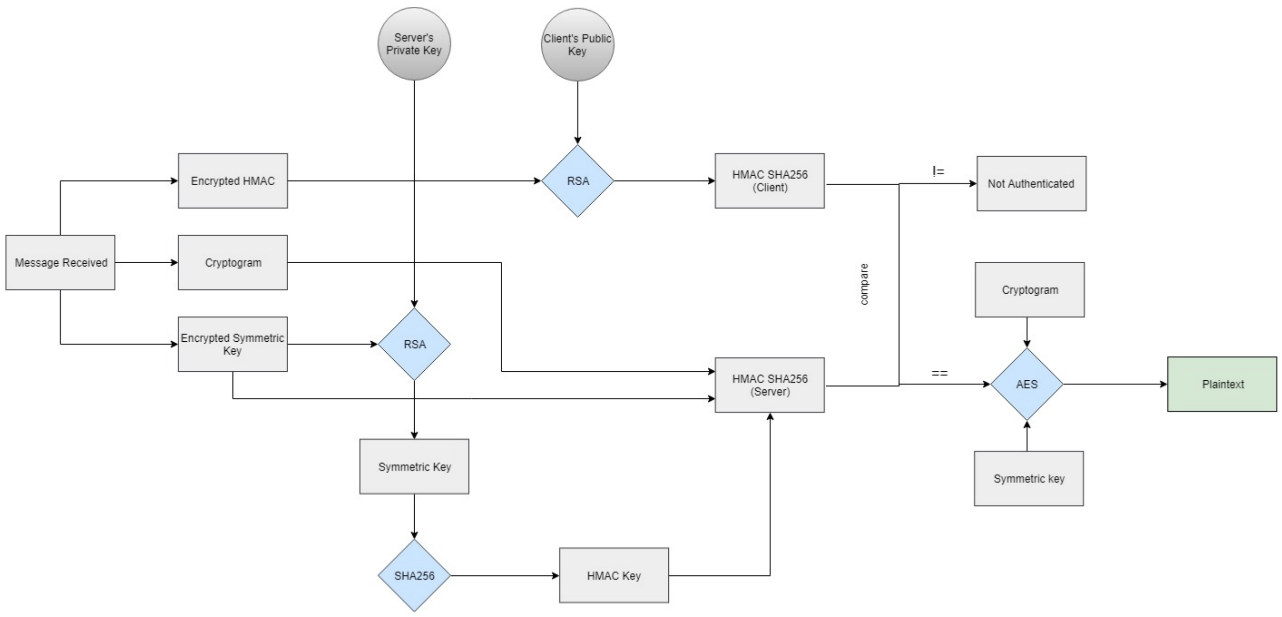

Figure 5: Message decryption diagram.

4.3 Storage Security

As we have seen before, all information exchanged will be secure while the memory is volatile. But we also need to secure it in the non-volatile memory. The best alternative, we have encountered is the use of the GNUPG (see Figure 2) for encrypting all the files with a master password for each server:

- On the Client's side (user), there will be receipts stored in files and in the Repository Server there will be a file containing the auction's data. Both files will be encrypted using GnuPG (GNU Privacy Guard).
- The user will calculate the MD5 of each receipt stored (see Figure 1). Every time one wants to use a receipt, one will calculate the MD5 of the current receipt and compare it to the pre-calculated MD5. If they do not match, the receipt might have been adulterated.
- The user will be authenticated by the system through a log in on the system's client interface. He/She must insert the Cartão de Cidadão in a card reader and everytime there is the need to sign messages the user must insert their PIN in order for the private key to be used.

Bibliography

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