Segurança 2018/2019

Sistema de Leilões

8240 - MESTRADO INTEGRADO EM ENGENHARIA

COMPUTADORES E TELEMÁTICA

Filipe Reis
NMec: 76414 filipereis96@ua.pt

Carlos Ribeiro
NMec: 71945
carlosfiliperibeiro@ua.pt

Grupo P3G2

16 November 2018

Conteúdos

1 Introdução

Foi nos proposto como projeto da cadeira de Segurança, realizar um sistema de gestão de leilões com blockchain. A segurança destes sistemas é muito importante, porque os clientes querem saber que um leilão foi justo, por exemplo quem ganhou um leilão foi o cliente que fez a licitação maior, que as bids não foram alteradas, ou que houve algum tipo de intruso. Para desenvolver um sistema seguro foram tomadas várias medidas de segurança que vão ser referidas nos pontos a seguir.

2 Requirementos

O sistema precisa:

- **Confidencialidade, integridade e autenticação de licitações** Os bids não podem ser modificados ou alterados de alguma maneira.
- **Aceitação e confirmação de licitações** Os bids apenas podem ser aceitados, caso estejam de acordo com um conjunto de parameteros.
- **Identidade e anonimato da identidade dos autores das licitações** Os bids estão ligados aos utilizadores, e devem permanecer sempre anonimos até ao fim de um leilão.
- **Certeza de honestidade** Apesar dos servidores, terem acesso a muitas informações sobre os bids, deve ser garantida a honestidade dos mesmos.

3 Componentes de um sistema

O sistema é baseado em 3 partes :

- **Auction Manager** Servidor para a manutenção dos bids.
- **Auction Repository** Servidor com toda a informação sobre sobre os leilões.
- **Clients** Cliente com acesso a um conjunto reservado de operações possíveis.

4 Mecanismos

4.1 Distribuição de chaves

Para o caso das chaves assimétricas era informado que cada servidor já possuia um par de chaves, no nosso caso foi escolhido RSA.

4.2 Chaves simétricas

Para o caso das chave simétricas é gerada uma chave com AES que depois é enviada, para os servidores.

4.3 Cifras

O nosso sistema uma para cifras assimatricas RSA, com recurso as chaves obtidas com os certificados. Já para o caso das cifras simétricas que são as mais usadas, é usado AES em modo GCM, foi escolhido AES, devido á sua alta compatibildiade e segurança, como modo foi usado GCM devido á sua alta performance e ao ataque á questão de autenticação que tambem é feita com o uso deste modo.

4.4 Autenticação

O caso da autenticação é resolvido através do uso de GCM, pelo que não é necessário mais nada para alem disto.

4.5 Assinaturas

As assinaturas foram feitas com SHA256 e RSA, havendo sido SHA 256 devido ao equilibrio entre performance e segurança.

4.6 Certificados

Os certificados a serem gerados, pertecem aos servidores, e estes devem gerar um self-signed certificate, para isto foi adaptada uma função encontrada na internet, estando presente no código a sua fonte.

5 Satisfação dos requirementos

5.1 Certificação de servidores e de clientes

Os clientes querem ter sempre a certeza que estão a trocar informações com o sistema desenvolvido e não com outro sistema a fazer passar-se pelo sistema implementado. Como tal, cada servidor fornece um certificado a cada cliente e entre servidores, fornecendo por consequência a sua chave pública. Para um leilão acontecer sem problemas quanto à identificação do cliente, é pedido ao cliente em algumas situações para se autenticar com o cartão de cidadão, é trocado o certificado do cartão do cidadão com os servidores para os servidores poderem confiar na autenticação do cliente. No final da troca de certificados, é possível trocar mensagens encriptadas entre todos os sistemas com recurso a chaves assimétricas e encriptação com RSA, no entanto, esta troca de mensagens é bastante mais lenta que a troca com encriptação simétrica, como nós queremos que a troca seja o mais rápida possível isto leva-nos ao seguinte ponto.

```
//Manda mensagem em plaintext para o repositorio a informar que quer iniciar a comunicação
String sendMsg = "{ \"Type\":init}";
JSONObject sendObj = new JSONObject(sendMsg);
InetAddress ServerIP = InetAddress.getByName("127.0.0.1");
int ServerPort = 9876;
byte[] sendbuffer;
sendbuffer = sendObj.toString().getBytes();
DatagramPacket sendPacket = new DatagramPacket(sendbuffer, sendbuffer.length,ServerIP ,ServerPort);
clientSocket.send(sendPacket);

//Repositorio responde com o certificado do servidor
byte[] receivebuffer = new byte[30000];
DatagramPacket receivePacket = new DatagramPacket(receivebuffer, receivebuffer.length);
clientSocket.receive(receivePacket);

byte[] certificateBytes = receivePacket.getData();
CertificateFactory certificateFactory = CertificateFactory.getInstance("X.509");
X509Certificate certificate = (X509Certificate)(certificateFactory.generateCertificate( new ByteArrayInputStream(certificateBytes)));
repositoryCert = certificate;

//Valida certificado, guarda-lo e envia certificado do cliente
try{
    certificate.checkValidity();
    repositoryKey = certificate.getPublicKey();
    messageRepositoryCert(clientSocket, SecurityClient.getCCCertificate());
}catch(CertificateExpiredException | CertificateNotYetValidException e){
    System.out.println("Certificate not valid ");
}
```

Figure 1: Código relativo á troca de certificados.

5.2 Troca de mensagens cifradas

Como visto anteriormente a maneira mais rápida de trocar mensagens cifradas é através de encriptação simétrica, para isso é necessário os sistemas acordarem uma chave simétrica, isto ocorre imediatamente após a troca de certificados, é gerada uma chave simétrica com AES, uma vez gerada, é necessário transmitir a mensagem para o seu par, para isso é então assinada e encriptada a chave simétrica, a assinatura é feita com SHA-256 que é um algoritmo de hash ainda não quebrado, tornando-se assim uma das hash functions mais fortes existentes, já a encriptação é feita com RSA com recurso à chave privada sendo depois no seu par verificada a assinatura e decriptada com a chave pública obtida através do certificado previamente trocado.

```
if(simetricKeyGen){
    try {
        secretKeyManager= KeyGenerator.getInstance("AES").generateKey();

        //Assinar chave
        MessageDigest messageDigest = MessageDigest.getInstance("SHA-256");
        byte[] digest = messageDigest.digest(secretKeyManager.getEncoded());

        //Encriptar
        byte[] symKey = SecurityClient.encryptMsg(digest,SecurityClient.getPrivateKey());

        //Envia chave simetrica
        Gson gson = new Gson();
        String json = ""+gson.toJson(symKey);
        String json2 = ""+gson.toJson(secretKeyManager.getEncoded());
        sendMsg = "{ \"Sym\":"+json+",\"Data\":"+json2+"}";
        sendObj = new JSONObject(sendMsg);
    } catch (NoSuchAlgorithmException ex) {
        System.out.println("Impossible to generate Symetric Key.");
    }
}
```

Figure 2: Produção de chaves simétricas.

Uma vez com as chaves simétricas acordadas, é usado AES em modo GCM(Galois/Counter Mode), foi escolhido este modo, devido à sua eficiência e performance e também devido a atacar já um novo problema, que é a necessidade de autenticação entre os sistemas. Apesar de difícil continua a ser possível ataques de brute-force de modo á tentativa de descobrimento da chave simétrica, de modo a reduzir ainda mais essa remota possibilidade, sempre que é encriptada uma mensagem é gerado um novo IV com o recurso a um CSPNRG (cryptographically secure pseudo-random number generator).

```java
/**
 * Encripta uma mensagem, usando chaves simetricas
 *
 * @param input mensagem a ser encriptada
 * @param keySym chave simetricas
 * @param initializationVector array para vertor de inicialização
 * @return mensagem encriptada
 */
static byte[] encryptMsgSym(byte[] input, Key keySym, byte[] initializationVector) throws IOException, GeneralSecurityException {
    Cipher cipher = Cipher.getInstance("AES/GCM/NoPadding");
    GCMParameterSpec ivSpec = new GCMParameterSpec(16 * Byte.SIZE, initializationVector);
    cipher.init(Cipher.ENCRYPT_MODE, keySym,ivSpec);
    byte[] output = cipher.doFinal(input);
    return output;
}

/**
 * Decripta uma mensagem, usando chaves simetricas
 *
 * @param input mensagem a ser encriptada
 * @param keySym chave simetricas
 * @param initializationVector vetor de inicialização
 * @return mensagem decriptada
 */
static byte[] decryptMsgSym(byte[] input, Key keySym, byte[] initializationVector) throws IOException, GeneralSecurityException {
    Cipher cipher = Cipher.getInstance("AES/GCM/NoPadding");
    //Definir IV
    GCMParameterSpec ivSpec = new GCMParameterSpec(16 * Byte.SIZE, initializationVector);
    cipher.init(Cipher.DECRYPT_MODE, keySym, ivSpec);
    byte[] output = cipher.doFinal(input);
    return output;

}
```

Figure 3: Código relativo á troca encriptação e decriptação simétrica.

5.3 Autenticação e controlo de integridade

Como já vimos anteriormente, a solução que vimos de modo a resolver este problema, foi a utilização de GCM.

5.4 Proteção dos bids até ao final do leilão

É extremamente importante manter a proteção dos bids até ao final do leilão, para isso quando o cliente deseja fazer um bid, ele procede à assinatura do mesmo e envia para o repositório, que por sua vez envia para o manager, o manager procede à validação do bid e uma vez validado encripta o bid com uma chave que apenas ele possui, fazendo com que apenas ele possa desencriptar o bid, e devolve o mesmo ao repositório, que por sua vez adicionar à lista de bids do leilão desejado. No fim do leilão, o repositório procede ao pedido de desencriptação do mesmo pelo manager, o manager faz-lo, e no fim envia ao repositório, que agora já com o bid desencriptado pode verificar a assinatura do bid, garantindo que não houve alguma alteração do mesmo, pois caso tivesse havido, a validação da assinatura falharia.

5.5 Identificação do autor de um bid através do seu CC

De modo a garantir a identificação e não repudiação de um bid, sempre que um cliente tem de fazer uma bid, ele deve assinar o mesmo, a assinatura é feita com o recurso a SHA256withRSA, sendo usada a chave privada do cliente para assinar, no fim do leilão as bids são verificadas de modo a garantir que não houve qualquer alteração do mesmo, o que nos leva ao seguinte ponto.

5.6 Exposição dos bids necessários no fim de um leilão

No fim de um leilão devem ser revelados todos os bids e seus autores para o caso de um leilão inglês e apenas os bids para o caso de um leilão cego, no momento da revelação dos bids, são então verificadas todas as assinaturas, sendo todas as assinaturas verificadas com sucesso, são então dependendo do tipo de leilão remetidos para o cliente as informações necessárias.

```
String dataFromSignature = ",\"AuctionID\":"+AuctionList.get(i).getAuctionID()+",\"Amount\":"+AuctionList.get(i)
//Validar assinatura
if(SecurityRepository.verifySign(signedMessage, dataFromSignature.getBytes(), clientCert.get(clientID-1))){
    if(AuctionList.get(i).isEnglishAuction()){
        toClient.append("Valor",AuctionList.get(i).getBids().get(j).getValue());
        toClient.append("Cliente",AuctionList.get(i).getBids().get(j).getClientID());
    }else{
        toClient.append("Valor",AuctionList.get(i).getBids().get(j).getValue());
    }
    //System.out.println("Bid validada");
}
```

Figure 4: Validação de bids.

5.7 Validação dos bids

A validação dos bids como já visto, é feita pelo manager, esta validação deve ser dinâmica, e inicialmente deveria ser validado o caso de um leilão já ter terminado ou não e as regras básicas relativas a um leilão, que no caso de ser um leilão inglês, apenas exige que o valor da bid seja maior que a maior bid já feita, já no caso de um leilão cego, são todos os valores de bids aceitos. A parte da validação das regras básicas dos leilões foi concluída com sucesso, no entanto por alguma falta de tempo não foi possível a realização de outras validações, quer sejam relacionadas com o tempo ou com permissões específicas para cada utilizador.

```
/**
 * Valida bids, neste momento apenas verifica se no caso de um leilão ingles a bid é maior que a anterior
 *
 * @param amount Valor da bid
 * @param signed bid assinada
 * @return String onde é especificada a bid validada e a sua encriptação ou a não validação da bid
 */
private static String validateBids(double amount, byte[] signed, String auctionType, double lastBid) throws IOException, GeneralSecurityException{
    //Fazer validações
    boolean validBid = true;
    if(auctionType.equals("English")){
        if(amount<=lastBid) validBid=false;
    }
    Gson gson = new Gson();

    String ret = "";
    if(validBid){
        byte[] signedEncrypted = SecurityManager.encryptMsgSym(signed, bidEncrypt, initializationVectorBid);
        String encrypted = ""+gson.toJson(signedEncrypted);
        ret = "{ \"Type\":\"ret\",\"Message\":Valid,\"Encrypted\":"+encrypted+"}";
    }
    else ret = "{ \"Type\":\"ret\",\"Message\":Invalid}";
    return ret;
}
```

Figure 5: Validação de bids.

5.8 Modificação de bids validados

É dada ao manager a possibilidade de encriptação ou desencriptação do bid, no entanto no fim do leilão deve ser possível provar que não houve alteração do mesmo por parte do manager, isto foi de novo feito através de assinaturas, uma bid assinada nunca pode ser alterada, ou será automaticamente invalidada, logo se o manager ou outra qualquer entidade mudar algo acerca da bid, isto faz automaticamente que a assinatura deixe de ser válida o que impossibilita a possibilidade de prova da não alteração dos bids no momento do término de um leilão.

5.9 Construção de um blockchain por leilão

Era pedido a construção de uma blockchain para as bids de cada leilão de modo a garantir, que as bids não eram trocadas de ordem, para a implementação disso foi para leilão criada uma lista ligada ordenada, onde no momento da adição de cada bid, era adicionada a hash da lista de bids antes da inserção da nova bid, o que invalida a troca das bids, pois cada nova inserção, vai "selar" inserção prévia.

```
JSONObject receivedValid = new JSONObject(ReceivedMsg);
String validBid = receivedValid.getString("Message");
if(validBid.equals("Valid")){
    for(int i=0; i<AuctionList.size();i++){
        if(AuctionList.get(i).getAuctionID() == msg2.getInt("AuctionID"))
        {
            //Adicionar á blockChain
            byte[] hash = SecurityRepository.hash(AuctionList.get(i).getBids().toString().getBytes());
            AuctionList.get(i).addChain(hash);

            Bid b = new Bid(msg2.getDouble("Amount"),msg2.getInt("ClientID"));
            AuctionList.get(i).addBid(b);
            JSONArray datasigned = receivedValid.getJSONArray("Encrypted");
            byte[] signed = gson.fromJson(datasigned.toString(), byte[].class); //Hash
            AuctionList.get(i).addBidSignedEncrypted(signed);
        }
    }
```

Figure 6: Produção da blockchain.

5.10 Criptopuzzles

Como em todos os sistemas os clientes podem fazer ações desnecessárias que ocupam os servidores, para esse problema foi implementado nas licitações os criptopuzzles. Os criptopuzzles fazem com que os clientes demorem mais tempo a fazer uma licitação com um proof of work. O proof of work consiste em descobrir um número random enviado pelo servidor, quando o cliente descobre o número envia de volta o resultado e caso seja realmente o número gerado, é lhe dado o direito de fazer uma licitação.

```
static byte[] puzzleSolve(byte[] puzzle) {
    byte[] solution = new byte[2];
    while(!Arrays.equals(solution, puzzle)){
        SecureRandom secRan = new SecureRandom();
        secRan.nextBytes(solution);
    }
    return solution;
}
```

Figure 7: Resolução de um puzzle enviado pelo servidor.

5.11 Recibos

Para manter a transparência do leilão é enviado um recibo por cada licitação, isto permite um cliente reclamar se houver algum problema na escolha do vencedor. O servidor ao receber uma licitação e depois da validação dessa, envia o recibo com assinado com a sua chave privada. O cliente pode verificar um recibo com a chave pública do servidor.

```
//Produção de um recibo
case "Receipt":
    String receiptText = "";
    String fileName;
    System.out.print("\nInsira o nome do ficheiro para guardar como recibo: ");
    fileName = br.readLine();

    int receiptAuctionID = rec.getInt("AuctionID");
    double receiptAmount = rec.getDouble("Amount");
    int receiptClientID = rec.getInt("ClientID");
    String signature = rec.getString("Sign");

    receiptText += "Id do Leilão: " + receiptAuctionID;
    receiptText += "\nQuantia: " + receiptAmount;
    receiptText += "\nId do Cliente: " + receiptClientID;
    receiptText += "\n\nAssinatura digital: " + signature;

    try {
        File newTextFile = new File(fileName + ".txt");

        FileWriter fw = new FileWriter(newTextFile);
        fw.write(receiptText);
        fw.close();

    } catch (IOException e) {
        e.printStackTrace();
    }
    break;
```

Figure 8: Produção de um recibo.

6 Conclusão

Com o desenvolvimento deste projeto foi possível pôr em prática os conhecimentos aprendidos na cadeira de Segurança. O objetivo principal foi concluído com sucesso, procedendo à implementação de políticas de segurança no sistema de gestão de leilões, no entanto devido á pressão de outras cadeiras, não foi possível otimizar o código nem fazer tudo o que havia sido planeado, sendo por exemplo a utilização do tempo uma das coisas que foi deixada para trás.