Segurança

Authors

Manuel Santos 79951 Nuno Félix 80330

***Abstract* - Este relatório descreve o desenvolvimento e demonstração do projeto de Segurança**

I. Introdução

O objetivo deste projeto foi desenvolver um sistema que permite utilizadores criar e participar num jogo de Copas. O sistema deve garantir identidade e autenticação dos jogadores, distribuição confidencial e aleatória de cartas, garantia de honestidade, evolução correta do jogo e consenso na contabilidade.

II. Funcionalidades

Como referido na introdução nós implementamos este sistema, um sistema que garante as seguintes funcionalidades:

Functionalities to implement:
- Protection (encryption, authentication, etc.) of the messages exchanged;
- Identification of users in a croupier with their Citizen Card;
- Set up of sessions between players and a croupier (using the Citizen Card);
- Set up of sessions between players (using the Citizen Card);
- Agreement to join a table (using the Citizen Card);
- Deck secure distribution protocol;
- Validation of the cards played during a game by each player;
- Protest against cheating;
- Possibility of cheating;
- Game accounting agreement;

III. Entrar numa mesa

O cliente quando se conecta, o servidor envia-lhe as mesas atuais e os jogadores que nelas se encontram. O cliente pode escolher uma mesa e passar para a fase de autenticação.

IV. Cartão de Cidadão

O cartão de cidadão é uma grande parte do projeto visto que é responsável por autenticar cada jogador ao croupier e aos outros jogadores.

Para interagir como o cartão de cidadão usamos a biblioteca PyKCS11. O cliente quando se conecta gera um par de chaves EC que vai enviar ao servidor no processo de autenticação. O cliente autentica-se colocando o PIN do cartão de cidadão e envia ao servidor o seu nome e informação necessária à sua autenticação assinada com a chave privada do cartão de cidadão. O servidor obtém a chave pública do cartão de cidadão através do certificado e verifica se a assinatura do cliente é válida. Se for válida o servidor sabe agora um novo par de chaves do cliente passa aos outros clientes para verificarem a identidade de quem está a entrar na mesa e conhecendo assim a chave pública EC do emissor.

V. Troca de Mensagens Seguras

A. Protocolo de comunicação

Nós usamos Socket.io para a comunicação, um protocolo que permite comunicação em tempo real bidirecional baseada em eventos. O Socket.io é nativo em javascript, por isso usamos uma biblioteca python-socketio que nos permitiu criar um servidor web-socket e vários clientes web-socket assíncronos. Trouxe algumas vantagens a nível de implementação, visto que facilita o processo de gestão de várias mesas, não necessitando assim de threads para gestão de sockets tcp normais.

B. Entre cliente e croupier

O cliente e o croupier vão comunicar usando mensagens assinadas. Ambos geram um par de chaves assimétricas usando o algoritmo de curvas elípticas e no processo de autenticação por cartão de cidadão o servidor e cliente vão trocar as suas chaves públicas.

Toda a comunicação cliente - croupier é assinada pelo emissor com a sua chave privada EC e validada pelo recetor.

C. Entre cliente e cliente

Após todos os jogadores serem autenticados pelo croupier e pelos restantes jogadores da mesa, são criados canais seguros entre pares de jogadores usando Diffie Hellman e curvas elípticas, onde cada cliente fornece aos outros uma chave pública assinada pelo mesmo e verificada pelo croupier antes de redistribuir pelos outros clientes. Um cliente para usar o canal seguro com outro irá encriptar a mensagem com uma shared key gerada através da sua chave privada e chave pública do cliente com quem pretende comunicar (ambas as chaves são criadas com o algoritmo de curvas elípticas), o cliente do outro lado para decriptar a mensagem irá usar a sua chave privada e a chave pública do cliente que lhe enviou a mensagem. Desta forma garantimos que apenas o jogador com quem pretendemos comunicar consegue visualizar o conteúdo da mensagem.

VI. Protocolo de Distribuição Segura do Deck

A. Cifrar o deck

A cifragem do deck começa com um deck com todas as cartas gerado pelo croupier. O croupier escolhe um jogador aleatório para passar o deck. Esse jogador encripta o deck com uma chave simétrica, baralha e passa o deck ao próximo jogador.

No total o deck vai ser encriptado 4 vezes para garantir que apenas depois de serem partilhadas todas as chaves é q os jogadores terão acesso ao conteúdo das cartas. O algoritmo de cifra simétrica usado na cifra do baralho pode variar entre os seguintes algoritmos, AES, Camellia, IDEA, CAST5, SEED e BlowFish. A escolha do algoritmo usado em cada mesa cabe em parte aos jogadores, que quando pretendem juntar se a uma mesa de jogo dão uma sugestão de um algoritmo. O servidor, para ser justo para todos os clientes, irá selecionar uma das sugestões dos jogadores para ser usada em todos os processos de cifra simétrica.

B. Distribuição do deck

Para a distribuição de cartas por todos os jogadores da mesa usamos os canais seguros criados entre eles. Depois de ser feita a encriptação das cartas do deck por todos os jogadores, um jogador random começa o processo de tirar uma carta. Este processo tem duas possíveis ações, a primeira o jogador tira uma carta do deck e a segunda o jogador troca N cartas da sua mão com cartas do deck. Neste processo o jogador escolhe um outro jogador random a quem vai passar o deck, usando o canal seguro que possui com esse jogador (diffie hellman).
Quando todos os jogadores verificarem que o deck está vazio, produzem um Bit Commitment e enviam para o croupier (que irá distribuir por todos os outros jogadores) para garantirem a integridade da sua mão inicial.

C. Decifrar a mão

Depois de todos os jogadores terem 13 cartas na mão, necessitam de todos as chaves simétricas usadas na encriptação do deck e a ordem pela qual foram usadas. Cada jogador irá usar as chaves na ordem inversa para a decriptação da sua mão. Estas chaves são passadas de jogador para jogador usando os canais seguros criados anteriormente.

VI. BIT COMMITMENT

O bit commitment é necessário para verificar a veracidade da mão. Nesta etapa o jogador possui 13 cartas cifradas e cada jogador irá gerar uma prova dessas mesmas cartas antes de lhe ser permitido decriptar a sua mão. Para isso, o jogador começa por gerar duas chaves de 128 bits, R1 e R2 e de seguida é calculada a prova, á qual chamamos de bit commitment. Para gerar o bit commitment usamos a expressão, bitC = hash(R1,R2,C), onde usamos a hash, SHA256 e em que C é o conjunto de cartas encriptadas que temos na mão inicialmente.

VII. JOGAR

Para um jogador jogar uma carta deverá esperar por uma mensagem do servidor a pedir para o mesmo jogar, e só depois enviar uma mensagem assinada com a carta que pretende jogar. Na primeira jogada de cada ronda o jogador com o 2 de paus é o primeiro a jogar.

O cliente pode ser executado em modo manual ou automático. O automático foi implementado para facilitar o teste do cliente.

VIII. VERIFICAÇÃO DE BATOTA

A. Possibilidade de fazer batota

Todos os jogadores podem fazer batota no decorrer do jogo. A jogada maliciosa será enviada para o servidor como qualquer outra, e cabe ao outros elementos da mesa verificarem a existência de batota.

B. Protestar jogada

No sua vez de jogar, cada jogador tem a possibilidade de reportar uma batota feita por outro jogador (e identificar o tipo de batota).

C. Validação das cartas jogadas por um jogador

O croupier apenas irá validar as cartas se algum jogador se pronunciar. No nosso jogo de copas existe a possibilidade de fazer três tipos diferentes de batota.

Jogar uma carta que não tem na mão

A mão inicial do jogador é secreta as restantes entidades da mesa e por isso é apenas possível verificar a sua veracidade através do bit commitment gerado pelo jogador. Um jogador pode acusar outro enviando uma mensagem ao servidor com a carta maliciosa e a identificação do jogador acusado. O servidor irá pedir ao jogador acusado o seu R2 e o C (cartas da sua mão inicial cifradas) e as chaves usadas inicialmente para encriptar o deck, com isto pode gerar um novo bitCommitment', que vai usar para comparar com o valor enviado no início do jogo pelo jogador em questão. Se bitCommitment == bitCommitment', então o servidor irá usar as chaves para decriptar a mão do jogador e verificar a presença ou não da carta dita de maliciosa.

O jogo termina após esta verificação, quer o jogador seja culpado ou não.

Jogar a mesma carta duas vezes

Neste caso o jogador é acusado de jogar pela segunda vez uma carta. O servidor para verificar este tipo de batota irá ver no histórico de jogadas, todas as cartas jogadas pelo jogador acusado e concluir se ele fez batota ou não.

Renúncia

Este tipo de batota significa que um jogador não assistiu ao naipe jogado pelo vencedor da mão anterior, mesmo possuindo pelo menos uma carta de naipe na mão. Para detetar este tipo de batota o servidor irá ter em consideração o aviso do jogador e ficar atento as jogadas futuras do jogador acusado. Se ele jogar uma carta do naipe pelo qual foi acusado de não jogar, o jogo termina

IX. GAME ACCOUNTING AGREEMENT

A. Decentralized Blockchain

Para a contabilidade segura e produção de recibos usamos uma blockchain descentralizada que nos permite criar um registo público confiável de todo o processo do jogo.
O cliente torna-se um nó com um registo local da blockchain.

1 Bloco = 1 Round
Cada cliente opera como um nó
Cada cliente tem uma lista de transações pendentes.
Cada cliente pode colocar um bloco

Durante o jogo, cada jogada feita por um cliente é assinada por esse cliente e validada pelos outros clientes que após a verificação cada nó adiciona a transação à sua lista de transações pendentes.

Quando o round acaba o servidor escolhe um jogador aleatório para colocar o bloco (25% chance de ser o escolhido).

O nó com direito de colocar o bloco propaga o bloco pelos outros nós que após verificação atualizam o seu registo local blockchain.

B. Blockchain Viewer

Fizemos um simples visualizador da blockchain que nos permite visualizar o estado da blockchain em cada croupier. No final de cada round depois de cada nó ter atualizado o seu registo local, os nós comunicam ao servidor o seu registo local de blockchain assinado para o servidor comunicar com o exterior.

Para verificarmos o registo local de cada nó podemos aceder a https://localhost:5000/chain/<croupier_id>/<node_sid>

Passando como argumento o id do croupier e o identificador do nó podemos verificar as cópias locais de cada nó da blockchain.

X. DESENVOLVIMENTO

A. Deployment

Durante o desenvolvimento deste projeto o grupo usou o Git como sistema de controlo de versões. Mais informações pode ser encontrada no repositório do projeto:
http://code.ua.pt/projects/security1920-g20
To deploy our project you need to install some prerequisites:
Install swig

```
sudo apt install swig
```

Instalar as dependências

```
pip3 install -r requirements.txt
```

Correr o servidor

```
python3 wsserver.py [number_of_croupiers]
```

Correr o cliente em modo automático

```
python3 wsclient.py 1
```

Correr o cliente manual

```
python3 wsclient.py
```

III. OBSERVAÇÕES

A. Problems and deficiencies

- a biblioteca de Socket.io disconecta terminais por estarem inativos por um certo período de tempo. Para contornar este problema recorremos à biblioteca aioconsole que permite criar um equivalente ao input() mas assíncrono.
- Por algum motivo assinar com o cartão de cidadão torna a parte de tirar cartas do baralho mais lenta, não conseguimos identificar o problema.
- O software do cartao de cidadao pode não funcionar corretamente e nem sempre conseguimos validar as cadeias de certificados. A solução foi verificar que o método dessa mesma validação funciona mas deixar o processo fora do fluxo da autenticação dos clientes com o CC.

III. CONCLUSÃO

O desenvolvimento deste projeto permitiu-nos melhorar os nossos conhecimentos de segurança. Percebemos que criar uma aplicação com estes requerimentos de segurança é um processo que requer grande conhecimento dos métodos criptográficos e grande atenção ao detalhe.

Conseguimos realizar as funcionalidades pretendidas e ainda alguns extras como a implementação de vários métodos de cifras e a realização do último ponto do projeto recorrendo a uma Blockchain.

REFERENCES

[1] https://cryptography.io/en/latest/
[2] https://python-socketio.readthedocs.io/en/latest/