Secure multiplayer Hearts Game

Paulo Vasconcelos 84987
Pedro Teixeira 84715

***Resumo* – O presente relatório apresenta o desenvolvimento de um jogo de cartas multijogador seguro. Foi escolhido o jogo de copas para exemplificar o funcionamento da aplicação.**

O relatório começa por descrever sucintamente a lógica do jogo desenvolvido. De seguida, são apresentadas as principais funcionalidades e principais alternativas consideradas, terminando com uma discussão das limitações conhecidas.

I. INTRODUÇÃO

O sistema desenvolvido permite jogar de forma segura um jogo de cartas de Copas. Baseia-se na existência de um servidor (um *croupier*) e 4 clientes (4 jogadores). Os jogadores devem possuir cartão de cidadão para jogarem, embora seja possível jogarem sem recurso ao cartão.

Cada jogador, após ligar-se a um cliente, pode 1) criar uma mesa de jogo ou 2) juntar-se a uma mesa de jogo pré-existente.

Após 4 jogadores se registarem num dado jogo num determinado *croupier*, o jogo pode iniciar-se. Os 4 jogadores escolhem as cartas que pretendem jogar. Jogadas inválidas são detectadas.

No fim do jogo, os jogadores são inquiridos sobre se pretendem acusar batota de algum jogador. Se um jogador acusar batota, o *croupier* apresentará essa indicação.

Por último, após o fim da fase de protesto contra batota, cada jogador obtém e regista o estado do jogo (ie. data, ID do jogo e pontuações do jogador), permitindo resolver futuras disputas sobre o resultado de um jogo.

A. *Ficheiros fornecidos*

Pacote	Conteúdo
Server	Lógica associada ao servidor/*croupier*
Client	Lógica associada ao cliente/jogador
security	Lógica associada ao uso do cartão de cidadão, cifras simétricas e assimétricas, utilitários para conversão de/para Privacy-Enhanced Mail (PEM) Testes unitários
security	Certificados da cadeira de certificação do certificado do cartão de cidadão

Tabela 1 – Ficheiros fornecidos

B. *Como executar o jogo*

O jogo desenvolvido foi desenvolvido utilizando Linux e Python 3.6 e implica a instalação dos pacotes Python descritos no ficheiro requirements.txt (`pip3 install -r requirements.txt`), para além do middleware do Cartão de Cidadão. A instalação dos pacotes Python pode implicar a instalação do Swig [1].

Para executar o servidor a partir da raiz do projecto:

```
python3 Server/server.py
```

Para executar cada cliente a partir da raiz do projecto:

```
python3 Client/client.py -v -i '<Server IP>'
-p <Server UDP Port> -u <Username>
```

C. *Detalhes de implementação*

As principais mensagens e acções durante a execução de um jogo estão descritas na Figura 1.

Todas as mensagens trocadas entre o servidor e os clientes, ou entre clients (com excepção no processo de distribuição de cartas), são trocadas através de UDP, num formato proprietário `[Message Type]Data` (e.g. `[TableCreationRequest]<password>`). Isto permite controlo total sobre o formato das mensagens e significa que mesmo numa situação em que mensagens não cifradas sejam interceptadas, não seja trivial a sua interpretação.

Com o objectivo de assiok

nar mensagens consideradas críticas, tal como descrito no capítulo II, o cartão de cidadão de cada jogador é utilizado. Se não estiver disponível um cartão de cidadão para um jogador, e como o sistema não suporta que 2 ou mais clientes usem em simultâneo o mesmo cartão de cidadão, para **efeitos de teste** a aplicação é capaz de utilizar como alternativa a cifra assimétrica Rivest-Shamir-Adleman (RSA) com *padding* PKCS#1 Optimal Asymmetric Encryption Padding (AEP).

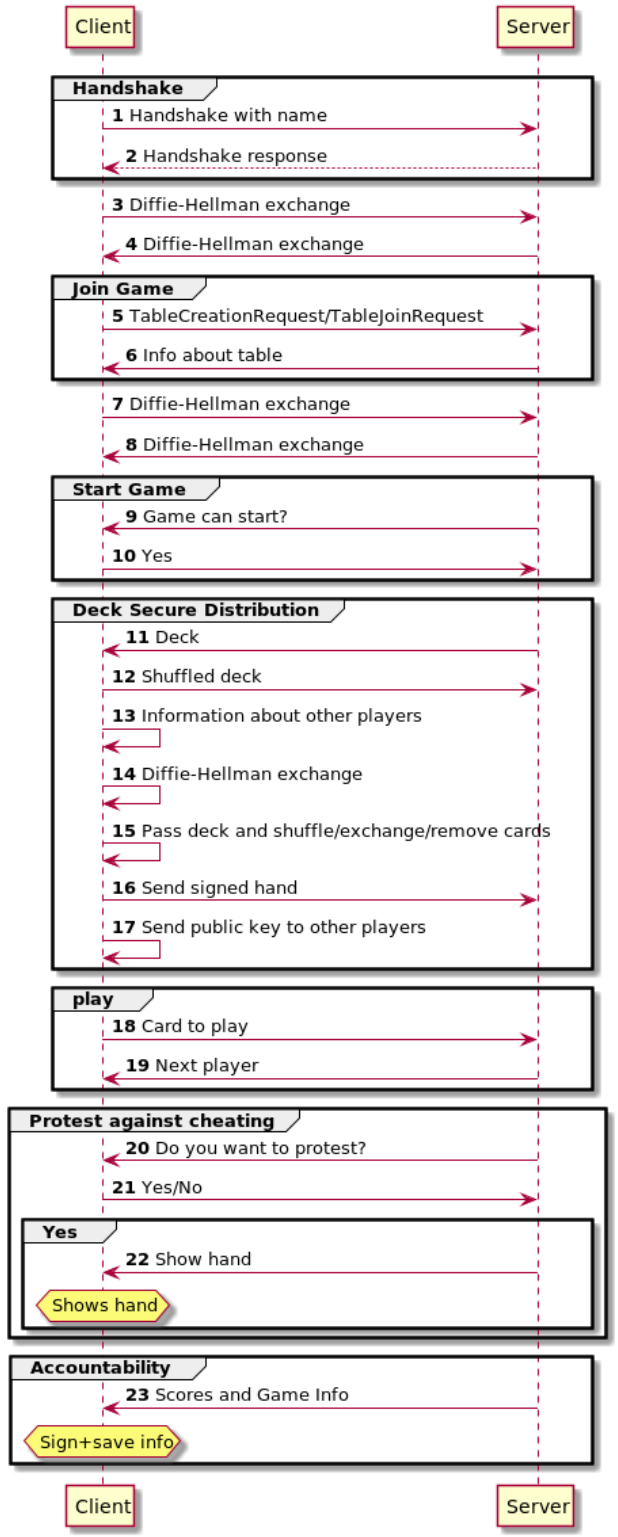

Figura 1 – Diagrama de sequência das principais acções e mensagens trocadas durante o jogo

II. Funcionalidades Desenvolvidas

A. *Protecção das mensagens trocadas*

Mensagens consideradas críticas (como a mensagem de criação/junção a uma mesa ou o término do jogo) foram assinadas utilizando o cartão do cidadão. Isto garante que a identidade do jogador que pretende jogar não pode ser falsificada.

As mensagens trocadas entre o *croupier*/servidor e utilizador/cliente ou entre clientes são cifradas e decifradas utilizando cifra simétricas Advanced Encryption Standard (AES) no modo Cipher Block Chaining (CBC), com um *Initialization Vector* aleatório e com um padding PKCS#7. Existe uma cifra simétrica para cada canal de comunicação entre cada cliente e o servidor e entre cada par de clientes.

A chave de cada cifra é gerada através de um processo de Password-Based Encryption (PBE), tendo sido utilizado o método de Diffie-Hellman para troca da palavra-passe que permite gerar a chave.

B. *Identificação dos utilizadores no croupier*

Quando um utilizador tem disponível um cartão de cidadão (que numa situação real, ie não de teste, é a única situação possível), é identificado e conhecido pelos outros jogadores e pelo *croupier* pelo nome presente no certificado de autenticação do cartão de cidadão. Isto torna impossível a falsificação da identidade do jogador.

Como, para efeitos de teste, foi necessário suportar a existência de um cliente sem cartão de cidadão, o cliente permite que seja o jogador a definir o nome pelo qual é conhecido, podendo até definir o mesmo nome presente num certificado do cartão de cidadão.

Esta possibilidade não deve ser considerada como uma vulnerabilidade, já que a possibilidade de os clientes não utilizarem o cartão de cidadão só existe para efeitos de teste e pela impossibilidade em termos materiais de testar 4 clientes com 4 cartões de cidadão.

C. *Set up de sessões entre jogadores e o croupier*

Quando um cliente inicia, é perguntado se pretende criar um jogo / mesa nova, ou se pretende juntar-se a uma mesa já existente. O cliente comunica a sua intenção ao *croupier* através das mensagens `[TableCreationRequest]` e `[TableJoinRequest]`, respectivamente.

Ambas as mensagens são assinadas pelo cliente utilizando o cartão de cidadão (se disponível) ou com a cifra RSA (se o cartão não estiver disponível), sendo a assinatura verificada pelo servidor.

Caso qualquer uma das assinaturas seja inválida, o jogo termina imediatamente.

D. *Set up de sessões entre jogadores*

Os dados trocados nas sessões entre os diferentes clientes/jogadores são cifrados e decifrados utilizando a cifra simétrica já referida na secção *A. Protecção das mensagens trocadas*.

Foi considerada e pensada a assinatura das mensagens trocadas entre os jogadores e a verificação das assinaturas. Contudo, isso implicava que cada cliente fornecesse o PIN do cartão de cidadão para cada mensagem ser assinada, diminuindo drasticamente a usabilidade do jogo (já a cada nova acção do jogador uma caixa de diálogo do *software* do cartão de cidadão apareceria).

E. *Acordo para entrar numa mesa*

Como referido na secção *C. Set up de sessões entre jogadores e o croupier*, um jogador comunica a sua intenção de entrar numa mesa, ou criando-a, ou entrando numa mesa já existente, através de mensagens que são assinadas. A assinatura é verificada pelo servidor e o jogo só continua se as assinaturas dos 4 clientes são válidas.

F. *Distribuição do baralho de cartas*

Para a distribuição do baralho de cartas, optou-se por um processo distribuído, em que os clientes recebem o baralho cifrado, cifrando-o e fazendo uma acção de entre 3 possíveis:

- O jogador pode baralhar o baralho;
- O jogador pode tirar uma carta do baralho, acrescentando portanto uma carta à sua mão;
- O jogador pode trocar uma carta que tenha na sua mão por uma carta que esteja no baralho;

Cada uma das acções é verificada, e nomeadamente na segunda e terceira possibilidades, é verificado se o jogador não tem a mão cheia e se o jogador ainda tem cartas na mão.

O processo de cifra é comulativo, ie. cada cliente, ao receber o baralho, cifra-o com a sua própria cifra, sendo que quando for necessário revelar as mãos o baralho tem de sofrer uma decifra pela ordem inversa, ie. $carta\ cifrada = C_1(C_2(C_3(C_4(carta\ original))))$ e $carta\ original = D_4(D_3(D_2(D_1(carta\ cifrada))))$, sendo 1, 2, 3 e 4 os 4 jogadores do jogo, C a função de cifragem e D a função de decifragem.

G. *Protesto contra batotas*

No fim do jogo, os jogadores são inquiridos sobre se pretendem acusar batota de algum jogador. Se um jogador acusar batota, o *croupier* apresentará essa indicação.

Se um jogador utilizar o mecanismo de protesto contra batotas, os jogadores são forçados a mostrar as suas cartas, permitindo a análise de potenciais batotas.

H. *Validação das cartas jogadas e possibilidade de batota*

No decorrer do jogo, um jogador pode jogar de forma ilícita de 2 formas diferentes:

- Jogando uma carta que não pode jogar tendo em conta as regras do jogo, neste caso copas;
- Jogar uma carta que não tem na sua mão (possível através do comando `\cheat` durante o jogo);

A possibilidade de jogadas ilícitas por parte de cada utilizador torna necessário que as cartas jogadas sofram algum tipo de validação.

I. *Accounting do jogo*

Após o fim da fase de protesto contra batota, cada jogador obtém as pontuações dos 4 jogadores, que serão guardadas num ficheiro.

Nesse ficheiro (gravado na pasta `accounting`), para além das pontuações, também são guardadas outras informações sobre o jogo (data e ID da mesa) e a assinatura dessas informações.

Futuras disputas sobre o resultado do jogo podem ser resolvidas verificando essa assinatura.

III. LIMITAÇÕES DA SOLUÇÃO

A solução final privilegiou o desenvolvimento de uma solução funcional e segura, mas também usável e com um desempenho computacional reduzido.

A decisão de não assinar todas as mensagens, seguida de uma verificação da validade da assinatura permitiu não só desenvolver uma solução mais eficiente sob o ponto de vista computacional, mas também um jogo usável. De facto, se todas as mensagens fossem assinadas, o *software* do cartão de cidadão iria exigir, através de uma caixa de diálogo, em cada mensagem a ser assinada, o PIN associado à *Citizen Authentication Key*. Tendo em conta o número de mensagens trocadas durante um jogo, tal impliciaria uma péssima usabilidade para o utilizador. Soluções para substituir a interface do *software* do cartão de cidadão na obtenção do PIN foram exploradas, mas sem sucesso.

Também a validação da cadeia de certificação foi tentada, sem sucesso. Após a obtenção dos certificados da cadeia de certificação (disponíveis em `security/certs`), foi desenvolvido um método de verificação da cadeia de certificação. Contudo, este não foi incorporado na lógica nem do cliente nem do servidor, uma vez que, até ao momento da entrega, não foi possível perceber o motivo do erro obtido nos testes unitários (*Unable to get issuer certificate*).

REFERÊNCIAS

[1] http://www.swig.org/download.html