universidade de aveiro

DOMINÓ MULTI-JOGADOR ONLINE SEGURO

47232 Segurança

Mestrado Integrado em Engenharia de Computadores e Telemática

2020-2021

Grupo

P18

Alunos

André Almeida 88960

Pedro Gonçalves 88859

Pedro Silva 89228

Rita Amante 89264

Professor

André Zúquete

Índice

1 Introdução

Este projeto foi desenvolvido no âmbito da Unidade Curricular de Segurança e consiste num sistema que permite que vários utilizadores joguem dominó, online.

O presente relatório pretende dar a conhecer o sistema referido. Para tal, no seu decorrer, serão clarificados os seguintes aspetos:

- Conceito do projeto, onde é feita a descrição do jogo;
- Funcionalidades implementadas, onde são descritos pormenorizadamente: a atribuição de pseudónimos a cada jogador, a comunicação segura, a distribuição segura do baralho, a validação das peças jogadas durante o jogo por cada jogador, a possibilidade de fazer batota, o protesto contra batota, a contagem do jogo e a escolha de peças do *stock* quando o jogo está parado;
- Especificações de segurança, onde são apresentadas as descrições dos algoritmos implementados;
- Execução do projeto, onde é descrito o que é preciso instalar e como é que se deve correr o projeto.

2 Conceito do projeto

Este projeto consiste num sistema que permite que vários utilizadores joguem dominó, online. O jogo é gerado automaticamente, no entanto, é preciso a interação do utilizador antes de se iniciar o jogo.

O jogo é constituído por quatro entidades: três clientes, que atuam como jogadores, e um servidor, que atua como gerente de mesa.

Quando se inicia o jogo no servidor, é definido o baralho do jogo, que é constituído por 28 peças. Cada peça é composta por duas pontas, cada uma com um número, por exemplo 3:4. A estas peças serão associados pseudónimos.

Quando um cliente entra no jogo é necessário estabelecer comunicação com o servidor, de modo a que o servidor e o cliente criem um canal de comunicação seguro entre os dois, sem que um terceiro consiga interpretar as mensagens transmitidas nesse canal. Depois de estabelecer comunicação, ao cliente será associado um pseudónimo, de forma a identificá-lo e para que nenhuma outra entidade o reconheça.

Após estabelecer a conexão, o cliente terá de inserir as credencias do seu cartão de cidadão. Para isso, o cliente recebe uma mensagem para inserir o cartão de cidadão e, quando o tiver feito, prime [Enter]. Seguidamente, terá de introduzir o pin de autenticação correto, de forma a assinar o pseudónimo. Após o processo estar concluído, este receberá uma mensagem informando que pode retirar o cartão com segurança.

Depois de estarem todas as entidades presentes (o servidor e os três clientes), o cliente *host*, ou seja, o primeiro cliente a entrar no jogo, terá de iniciar o jogo. Inicialmente, cada jogador escolhe 5 peças do baralho.

Para um jogador efetuar uma jogada é preciso que este contenha uma peça que tenha um dos valores da extremidade da mesa. Imaginemos que o jogador tem na mão as seguintes peças [3:1, 4:3, 5:0] e a mesa está com [2:1; … ; 5:4], então o jogador jogará a peça 4:3. Caso o jogador não tenha nenhuma peça que se enquadre à jogada e não haja nenhuma peça no *stock*, ele passa a vez ao próximo jogador.

O jogo termina quando:

- Há um vencedor, ou seja, quando um dos jogadores ficam sem peças na mão;
- Há um empate, ou seja, quando já não há peças no baralho e os jogadores não têm peças para assistir à jogada que está na mesa;
- É detetada batota, ou seja, quando é detetada uma jogada de uma peça feita por um jogador que não possua essa peça, ou quando algum jogador mente na sua pontuação final.

3 Funcionalidades implementadas

3.1 Atribuição de pseudónimos a cada jogador

3.1.1 Pseudonimização dos jogadores

O processo de pseudonimização dos jogadores inclui a importante funcionalidade de segurança de assegurar que o jogador que se ligou ao pseudónimo X no início do jogo é o mesmo que recebe os pontos que ganhou.

No fim do jogo, todos os pseudónimos são "queimados" de forma a nunca mais poderem ser usados em qualquer jogo futuro.

Ao ser declarado o vencedor, para além do processo de confirmação da identidade do vencedor, é adicionado a um registo os pontos deste no final do jogo.

3.1.2 Escolha dos pseudónimos

Para cada jogador será feito o seguinte procedimento:

- *Prompt* para o jogador escolher o seu pseudónimo;
- Verificação de que o pseudónimo nunca foi utilizado num jogo anterior;
- Dado que o cartão de cidadão do jogador esteja introduzido num leitor:
    - *Prompt* do pin de autenticação;
    - Se o pin foi introduzido corretamente, será feita a assinatura digital do pseudónimo com a chave privada da autenticação obtida da introdução do pin;
    - O pseudónimo e a assinatura associada serão armazenados num dicionário, no formato pseudónimo -> assinatura(pseudónimo);
    - O pseudónimo e a chave pública associada ao dono deste serão também armazenados num dicionário, no formato: pseudónimo -> chave pública;
    - Nos últimos 2 pontos, os dicionários mencionados estão guardados no ficheiro *game_data.json*, englobados num único dicionário.

Nota: O jogador receberá um *prompt "*`Introduce CC in reader. [Enter]`*"* e só deverá retirar o cartão quando aparecer a mensagem *""*`CC" can be removed safely`*"*.

3.1.3 Processo de "queima" dos pseudónimos

O ficheiro *used_pseudonyms.txt* guarda todos os pseudónimos usados em todo e qualquer jogo, um em cada linha. No fim dos jogos, todos os pseudónimos usados nesse jogo serão adicionados a esse ficheiro.

3.2 Comunicação segura

3.2.1 Comunicação Cliente-Servidor

A comunicação entre dois processos consiste na transmissão de uma mensagem de um socket num processo para um socket noutro processo. De modo a estabelecer uma comunicação entre o jogador e o gerente de mesa, criou-se uma sessão TCP, onde a comunicação usa uma abstração *stream* para ler e escrever dados. Quando dois processos tentam estabelecer uma ligação através de Sockets TCP, um desempenha o papel de cliente e outro de servidor.

O cliente cria um objeto do tipo Socket que tenta estabelecer uma ligação com um porto de um servidor, numa máquina remota, precisando de indicar o endereço IP e o porto da máquina remota. O servidor cria um objeto do tipo "listening socket" associado ao porto servidor. Este socket possui um método que fica bloqueado até que receba um pedido de ligação ao porto correspondente e, quando chega o pedido de ligação, o servidor aceita-o instanciando um novo socket que, tal como o socket do cliente, tem duas streams associadas, uma para saída outra para entrada de dados.

Após o cliente estabelecer uma sessão TCP com o servidor, estes têm de partilhar o mesmo segredo para conseguirem comunicar de forma segura, ou seja, sem que outro cliente consiga ver as mensagens trocadas entre ambos. Assim, para estabelecer a comunicação entre o jogador e o gerente de mesa, implementou-se o algoritmo de Diffie-Hellman (secção 4.1).

Deste modo, a ligação cliente-servidor estabelece-se da seguinte forma:

1. O cliente e o servidor obtêm dois números primos $g=11$ e $p=593$;
2. O cliente e o servidor geram as suas chaves privadas a e b, respetivamente, aleatoriamente, através da função *random.randint(10000, 999999)*);
3. O cliente e o servidor geram as suas chaves públicas A e B, respetivamente, onde $A=g^a \mod p$ e $B=g^b \mod p$. Depois, enviam-na um ao outro;
4. Depois do cliente receber a chave pública do servidor e este receber a chave pública do cliente, ambos calculam o segredo partilhado. O cliente calcula $K_A = B^a \mod p$ e o servidor calcula $K_B = A^b \mod p$;
5. O servidor depois de calcular o segredo, cifra uma mensagem de status "Online", segundo o algoritmo de criptografia AES, e envia-a para o cliente;
6. Quando o cliente receber a mensagem "Online" cifra vinda do servidor, decifra-a e, caso a mensagem decifra corresponda à mensagem "Online", envia uma mensagem de resposta "OK" cifrada, segundo uma segundo o algoritmo de criptografia AES, para o servidor;
7. Quando o servidor receber a mensagem "OK" cifrada vinda do cliente, decifra-a e, caso a mensagem decifra corresponda à mensagem "OK", a ligação fica estabelecida.

3.2.2 Comunicação Cliente-Cliente

Uma vez que um cliente não pode comunicar diretamente com outro cliente, a comunicação cliente-cliente tem de ser gerida pelo servidor.

A comunicação cliente-cliente está presente antes do jogo ser iniciado, esperando pela informação (enviada pelo servidor) de todos os participantes envolvidos no jogo. Aquando o início do jogo, inicia-se uma comunicação cliente-cliente, onde o cliente fonte envia uma mensagem ao servidor a indicar com quem é que pretende comunicar, que lhe responde com uma mensagem de confirmação de receção. Após o cliente receber a confirmação do servidor, o cliente começa a negociar o algoritmo Diffie-Hellman (secção 4.1) para estabelecer comunicação com o outro cliente.

Na negociação do algoritmo Diffie-Hellman, o cliente fonte obtém dois números primos *g* e *p*, gera a sua chave privada *a*, calcula a sua chave pública $A=g^a \mod p$ e envia-a para o servidor. O servidor recebe a mensagem e envia-a diretamente para o cliente destino. Do outro lado, o cliente destino recebe a mensagem, gera a sua chave privada *b*, calcula a sua chave pública $B=g^b \mod p$, calcula o segredo partilhado com a chave pública recebida do cliente fonte e envia a sua chave pública para o servidor, que envia diretamente para o outro cliente. Depois do cliente fonte receber a mensagem do servidor, calcula o segredo partilhado e envia uma mensagem cifrada com o segredo calculado para o servidor, que, mais uma vez, envia diretamente para o cliente destino. O cliente destino recebe a mensagem do servidor vinda do cliente fonte, decifra-a e fecha a ligação, enviando uma mensagem para o servidor.

A comunicação cliente-cliente ocorre também durante o jogo, onde são enviadas as listas de pseudónimos escolhidos e de pseudónimos por escolher entre jogadores na fase de seleção. Deste modo, o servidor inicialmente envia para todos os clientes as chaves públicas dos clientes envolvidos no jogo. Um cliente ao receber as chaves públicas irá verificar quais são as que não correspondem à dele próprio e estabelece um segredo com os outros dois clientes, usando as chaves públicas achadas. Depois, sempre que querem enviar uma mensagem para outro cliente, cifra com o segredo já calculado, que está guardado numa variável.

3.3 Distribuição segura do baralho

De forma a que o baralho seja corretamente distribuído, são precisas várias fases, pois é essencial que o servidor não saiba quais são as peças que cada cliente possui. No final da distribuição, o servidor saberá quais são as 15 peças que os 3 clientes possuem, mas não saberá que peça está em que jogador.

3.3.1 Fase de pseudonomização

Esta fase dá-se quando o baralho é criado. O servidor ao ser inicializado cria um objeto da classe *Game*, que possui um atributo *deck,* que inicializa a classe *Deck.* A classe deck cria as 28 peças, que são lidas de um ficheiro de texto *pieces*.

Para cada peça, existe uma função hash, que através de um número aleatório e do valor das suas subpeças, devolve um número. Após a criação das 28 peças, será ordenado numa lista o número originado por cada peça pela função hash. A partir dessa lista, será dado o índice da lista do número criado pela hash ao pseudónimo da peça correspondente. Através desse pseudónimo, é criado um baralho pseudonomizado, chamado *pseu_deck* que possui todos os pseudónimos.

3.3.2 Fase da randomização

Nesta fase, caraterizada pela ação *rcv_deck_to_encrypt*, o servidor manda o baralho pseudonomizado para o primeiro cliente, que cria uma chave aleatória para cada pseudónimo e cifra esse pseudónimo. Essa cifragem é feita com a função *encrypt_aes_pycrypto,* (secção 4.2). Esse mesmo cliente guardará, num dicionário, uma associação entre o pseudónimo e a chave usada para cifrar. Esse cliente, no final, enviará uma lista baralhada com todas as cifras resultantes para o servidor, que encaminhará para o próximo cliente. Esse cliente fará exatamente o mesmo processo que o primeiro cliente, recebendo, em vez dos pseudónimos, as cifras do cliente anterior. O último cliente a realizar este processo no final manda um sinal ao servidor para passar à próxima fase.

3.3.3 Fase de seleção

Nesta fase, são enviadas 2 listas entre os jogadores, uma lista com 28 elementos inicialmente, numerados de 1 a 28, representando a ordem das cifras devolvidas pelo último cliente ao servidor, e a outra lista com os elementos já escolhidos, que obviamente, irá começar vazia.

Cada cliente pode então escolher 1 de 3 opções, escolher um elemento da lista que representa a ordem das cifras do baralho cifrado, trocar um elemento que já possui por um elemento da lista do baralho cifrado, caso já tenha escolhido algum elemento previamente, ou passar ao próximo jogador sem mexer em nenhuma das listas. O cliente guarda os seus elementos escolhidos numa lista chamada *picked_pieces*. Quando o cliente troca um elemento, retira-o da lista dos elementos já escolhidos e da lista *picked_pieces*, coloca-o na lista de elementos por escolher, retira um elemento da lista de elementos por escolher e coloca esse

elemento na lista de elementos escolhidos e na lista *picked_pieces.* No final, cifra estas listas com a chave partilhada com o cliente imediatamente a seguir, para que o servidor não saiba que opção o jogador escolheu, e envia para o servidor, que depois irá enviar para o próximo cliente. Quando a lista dos elementos escolhidos possuir 15 elementos, avança para a próxima fase.

3.3.4 Fase de preparação para a desanonimização das peças

Optámos por realizar esta fase antes da fase de revelação, que consiste em criar uma lista onde cada cliente coloca as suas chaves públicas no índice correspondente às suas peças. O cliente recebe previamente uma lista com os elementos escolhidos e uma lista com as chaves públicas de cada peça, que para o primeiro cliente encontra-se vazia. Compara esses elementos com os que possui na sua lista *picked_pieces,* criando uma chave pública através do algoritmo RSA, que gera um par de chaves assimétricas, por cada elemento e colocando-a na lista de chaves públicas.

3.3.5 Fase de revelação

Nesta fase, o último cliente a cifrar o baralho receberá uma lista com os elementos escolhidos e que terá de decifrar. Essa lista possui 15 elementos numerados com números entre 1 e 28, correspondendo cada elemento ao índice da lista de peças cifradas – 1. Esse cliente devolverá uma lista com 15 elementos decifrados e uma lista com as chaves pública que serão enviadas para o segundo cliente que cifrou o baralho, cifradas com a sua chave secreta partilhada. Ao chegar ao último cliente, este devolverá uma lista com os pseudónimos e a lista de chaves públicas ao servidor, que pela primeira vez terá acesso às chaves públicas. Assim, o servidor desconhece de quem são as chaves públicas.

3.3.6 Fase de desanonimização das peças

Nesta última fase antes de poder começar realmente o jogo, o servidor associa cada peça ao pseudónimo correspondente, passa-a para bytes, e cifra-a com a chave pública associada. No final, envia uma lista com 15 peças cifradas para todos os jogadores. Os jogadores recebem todas as peças cifradas e verificam quais conseguem decifrar com as chaves privadas que possuem. Se conseguirem decifrar, adicionam essa peça à sua mão.

3.4 Validação das peças jogadas durante o jogo por cada jogador

Relativamente à validação das peças jogadas durante o jogo, o servidor começa por verificar se há peças na mesa, se não houver em princípio a peça jogada vai ser válida (o jogador pode logo fazer batota na primeira jogada, entraremos em detalhe no capítulo de Protesto contra batota).

Caso já haja peças na mesa, o servidor verifica quais os valores da metade das peças dos cantos da mesa onde dá para jogar e compara com a peça jogada, se forem o mesmo número e a peça tiver sido posicionada da forma correta, a jogada é então considerada válida.

3.5 Possibilidade de fazer batota

Por vezes é possível um cliente fazer batota para tentar ter o melhor resultado possível no jogo.

O jogador pode fazer batota em 2 ocasiões:

- Quando não possui uma peça para jogar, isto é, quando os valores das pontas das peças que estão na mesa não correspondem aos valores das peças que tem na sua mão. Isto faz com que tenha de apanhar uma peça do *stock,* ou que invente que possui uma peça que contém um dos valores das pontas da mesa. A decisão entre apanhar uma peça do *stock* ou fazer batota, é dada por um número aleatório;
- Quando no fim do jogo, o jogador possui muitos pontos na mão, e para que os outros não possam tirar partido disso e ficar com muitos pontos na sua identidade, mente no número de pontos que possui. O número de pontos que finge possuir é dado por um número aleatório baixo.

Obviamente isto terá consequências no jogo, que serão abordadas a seguir.

3.6 Protesto contra batota

Quando algum jogador faz batota, os outros saem prejudicados. Logo o servidor, como é honesto, poderá detetar uma jogada inválida, verificando se a peça está na mesa ou no *stock*, obrigando o mesmo jogador a jogar uma peça diferente. Caso o jogador faça uma jogada válida, mas de uma peça que está na mão de outro jogador o jogo continuará.

Caso um jogador detete na mesa uma peça que está na sua mão, pode protestar e dizer ao servidor que alguém jogou uma peça que ele possui. A partir daí o jogo acaba.

Quando um jogador mente na sua pontuação, será apanhado pelo servidor que faz uma contagem dos pontos, baseando-se nas peças que estão no *stock*, nas peças que estão na mesa e nas peças do *stock*. A partir desses pontos, sabe quantos pontos estão na mão dos 3 jogadores. Se a soma do resultado da mão dos 3 jogadores coincidir com a contagem de pontos do servidor, nenhum jogador está a mentir. Se não coincidir, o jogo acaba e não deixa ninguém colocar os pontos na sua identidade.

3.7 Contagem do jogo

3.7.1 Contagem de pontos num jogo

A contagem de pontos num jogo é feita através do valor de cada peça. Cada peça é avaliada pelos seus números, onde a peça [0:0] vale 0 pontos, a peça [2:1] vale 3 pontos e a peça [6:6] vale 12 pontos.

No fim do jogo, cada jogador envia a sua pontuação, baseada nas peças que possui na sua mão nesse momento, para o servidor. Quando o servidor tiver todas as pontuações, verifica se estas estão corretas e envia todas as pontuações para todos os jogadores. Assim, cada jogador saberá também a pontuação dos outros jogadores. No fim, a pontuação final de um jogador é dada através da subtração da pontuação do jogador que fez mais pontos pela pontuação do próprio jogador. Desta maneira, o jogador que ficou com mais pontos durante o jogo ficará com uma pontuação final de 0 pontos, ou seja, não pode adicionar pontos à sua identidade.

3.7.2 Atribuição dos pontos ao vencedor

O vencedor terá a escolha de atribuir os pontos ao seu registo total. Caso não o pretenda fazer, esta etapa é saltada. Caso contrário, o seguinte processo é efetuado:

- O programa obtém a chave pública da autenticação do cartão de cidadão do dicionário que guarda as chaves públicas;
- Com essa chave, é feita a verificação da assinatura (pseudónimo) de forma a assegurar que a pessoa que escolheu o pseudónimo é a mesma a que será atribuída os pontos;
- Dado o sucesso da etapa anterior, é agora atualizado o registo de pontos da identidade vencedora:
    - identity_points.csv, que guarda em formato csv, em cada linha:
        - Uma string do identificador da pessoa, retirada do cartão de cidadão;
            - Esse identificador consiste numa série de números associados ao certificado da chave pública da autenticação do CC;
        - A soma de todos os pontos ganhos por essa pessoa na totalidade dos jogos.

3.8 Escolha de peças do *stock*

A escolha de peças do *stock* dá-se apenas quando um jogador não tem peças para fazer uma jogada válida e não faz batota.

Para apanhar uma peça do *stock*, o jogador receberá os restantes elementos das peças que ainda não foram escolhidas, 13 inicialmente pois 15 já foram escolhidas pelos 3 clientes no início do jogo, que correspondem à lista de cifras devolvida pelo último cliente a cifrar o baralho. Após escolher um desses elementos, esse elemento será decifrado pelos 3 clientes pela ordem inversa pela qual foi cifrado.

No final, o primeiro jogador que cifrou o baralho devolverá o pseudónimo da peça ao servidor, que o usará para associar à peça que devolverá ao jogador que fez o pedido. Desta forma, o servidor saberá a peça que o jogador possui, mas não pode intervir nas peças que ele vai escolher, pois estas estão cifradas pelos 3 clientes.

4 Especificações de segurança

4.1 Algoritmo Diffie-Hellman

A troca de chaves de Diffie-Hellman é um método de criptografia específico para troca de chaves, que permite que duas partes que não possuam conhecimento a priori de cada uma, partilhem uma chave secreta sob um canal de comunicação inseguro. Tal chave pode ser usada para cifrar mensagens posteriores usando um esquema de cifra de chave simétrica.

Imaginemos que as entidades Alice e Bob querem trocar e-mails secretos de forma segura. No entanto, estas entidades nunca tiveram oportunidade de combinar uma PSK (*Pre-Shared key*) de forma segura. Para tal, a Alice enviou o seguinte e-mail ao Bob a explicar-lhe como é que vão trocar e-mails secretos de forma segura:

> Bob, vamos utilizar o algoritmo Diffie-Hellman para trocar uma chave pública. Esta chave, combinada com um algoritmo de criptografia AES (*Advanced Encryption Standard)* de 256 bits, fará com que o conteúdo dos nossos e-mails sejam cifrados. Assim, só quem conhece a chave privada (que será gerada pelo algoritmo Diffie-Hellman) poderá ler o e-mail (ou seja, só nós). Vou te explicar então como funciona o algoritmo Diffie-Hellman:
>
> 1. Eu irei escolher dois números que utilizaremos em comum. A = 7 e B = 11.
> 2. Escolho um número que só eu sei, que chamarei de X. Tu também escolherás um número (não podes contar a ninguém), chamado de Y.
> 3. Eu aplico a seguinte fórmula: $R_{alice} = (A^X) \bmod B$. E vou enviar-te o resultado de R_{alice}.
> 4. Tu aplicas a mesma fórmula, substituindo X pelo teu número secreto Y, ou seja, aplicarás $R_{bob} = (A^Y) \bmod B$. E também me enviarás o resultado de R_{bob}.
> 5. Depois, eu aplico a seguinte fórmula para descobrir nossa chave privada: $PSK = (R_{bob}^X) \bmod B$.
> 6. Para tu também descobrires a chave privada partilhada, usa a mesma fórmula, substituindo X por Y e R_{bob} por R_{alice}: $PSK = (R_{alice}^Y) \bmod B$.
> 7. Agora, é só trocarmos e-mails cifrados por AES-256, utilizando a chave PSK.
>
> Como disse no início, só nós é que conseguiremos decifrar o e-mail e ler o conteúdo pois só somos os únicos que temos conhecimento da chave PSK.

4.2 Cifras assimétricas e simétricas

As cifras assimétricas e a geração do par de chaves foram realizadas com recurso ao módulo pip cryptography. A cifragem simétrica, por outro lado, foi com base no package pycrypto.

4.2.1 Criptografia de chaves assimétricas

Geração de chaves

Realizado numa função no ficheiro asym_keys.py, a geração do par de chaves é realizada na função *generate_keys*, recebendo como argumento o tamanho em bits, que poderá ser 1, 2, 3 ou 4 Kib (1 Kib = 2^{10} bits).

O expoente público das chaves é 65527.

Cifragem / Decifragem

Realizado numa função no ficheiro ciphers.py chamada *encrypt_rsa_hazmat*, realiza cifragem com as seguintes características:

- Algoritmo de cifra/decifra é RSA;
- *Padding* usado foi OAEP (Optimized Asymmetric Encryption Padding);
    - *Hash* usada no padding foi SHA256

Assinatura/ Verificação

Realizados com as mesmas características da cifragem/decifragem com a exceção do *padding*, que é o PSS (*Probabilistic Signature Scheme*).

4.2.2 Criptografia de chaves simétricas

Cifragem / Decifragem

Estes dois processos são realizados com o algoritmo AES, sem padding (são adicionados espaços: “ ”, até a mensagem ficar alinhada.

5 Execução do projeto

O projeto foi desenvolvido em *Python* 3 e para o bom funcionamento do jogo, é necessária a instalação de vários pacotes:

- Python (sudo apt install python3.8);
- Cryptography (python3 -m pip install [-U] cryptography);
- Pycrypto (python3 –m pip install [–U] pycrypto);
- PyKCS11 (pip3 install PyKCS11).

Para executar o jogo, é necessário abrir quatro terminais, onde um corresponderá ao servidor e os restantes aos clientes. Em primeiro lugar, é preciso correr o servidor escrevendo na linha de comandos “python3 server.py”. Depois do servidor estar a correr é que se pode correr os clientes, escrevendo na linha de comandos “python3 client.py” em cada terminal.

Depois o servidor e os clientes estarem a correr, será pedido aos clientes que insira as credenciais do seu cartão de cidadão. O cliente recebe uma mensagem para inserir o cartão de cidadão e, quando o tiver feito, prime [Enter]. Depois, terá de introduzir o pin de autenticação correto e, após o processo estar concluído, este receberá uma mensagem a informar que pode retirar o cartão com segurança.

Depois, para iniciar o jogo, será apresentada uma mensagem no cliente host, ou seja, no primeiro cliente que entrou no jogo, para premir [Enter] e começar o jogo. Quando o jogo acabar, serão apresentadas informações relevantes do jogo, por exemplo, quem foi o vencedor.

6 Conclusões

Neste projeto foram implementadas todas as funcionalidades previstas à exceção do *bit commitment*. Por esta razão, a deteção de batota também está ligeiramente comprometida, apesar de funcionar por completo, tanto no servidor como no cliente, não há provas em relação a qual dos clientes é que fez batota.

Por vezes o jogo também poderá encravar em certas partes devido a *bugs* do *python* ou algum ruído na comunicação entre servidor e cliente. No caso de isso acontecer, terá de ser reiniciado o jogo.

Achámos particularmente difícil a cifragem do deck, no qual não podíamos cifrar a lista de chaves e pseudónimos como um todo, mas sim cifar cada uma das chaves públicas e pseudónimos de cada vez.

É de salientar que o jogo não foi implementado de raiz, tendo sido utilizado o código base do aluno Filipe Vale.

Para um trabalho futuro, poderiam ser integradas as funcionalidades que não conseguimos, bem como a resolução de alguns dos *bugs*, no entanto consideramos que foi um bom trabalho e ficámos bastante satisfeitos com o resultado.

7 Referências

- http://wiki.icmc.usp.br/images/d/d3/Rc05-aplicacao.pdf
- https://mathworld.wolfram.com/Diffie-HellmanProtocol.html
- https://github.com/FilipeMiguelVale/Secure-Domino
- https://steelkiwi.com/blog/working-tcp-sockets/
- https://docs.python.org/3/howto/sockets.html
- https://stackoverflow.com/questions/1157106/remove-all-occurrences-of-a-value-from-a-list