Dário Matos - 89288
Pedro Almeida - 89205
Pedro Valente - 88858
Samuel Duarte - 89222

Secure, multi-player online domino game

RELATÓRIO

"One single vulnerability is all
an attacker needs."

— Window Snyder

Resumo

O objetivo do projeto é desenvolver um sistema onde utilizadores poderão criar e participar num jogo de dominó. Este sistema é constituído por um número fixo de clientes e por um supervisor de mesa, com a particularidade de ter sido desenvolvido de forma a que a prioridade máxima fosse a segurança do mesmo.

Conteúdo

Lista de Figuras

Capítulo 1

Introdução

O objetivo do projeto é desenvolver um sistema onde utilizadores poderão criar e participar num jogo de dominó. Este sistema é constituído por um número fixo de clientes e por um supervisor de mesa, com a particularidade de ter sido desenvolvido de forma a que a prioridade máxima fosse a segurança do mesmo. Assim, as sessões estabelecidas entre o *Table Manager* (servidor) e os vários *Players* (clientes) são encriptadas e foi usado o protocolo Diffie-Hellman para a troca segura de valores para geração de chaves criptográficas acordadas entre duas entidades. Foi também implementado um protocolo de distribuição de peças seguro e métodos para verificar e detetar casos de batota.

Nos capítulos seguintes seguintes serão explicadas com mais detalhe as várias fases de desenvolvimento do projeto.

Capítulo 2

Estudos e alternativas

2.1 Estudos realizados e alternativas consideradas

A base teórica para a realização deste projeto incide na documentação da cadeira de Segurança. Foram consideradas e implementadas algumas alternativas no que toca ao sistema de pontos por jogador e à utilização de chaves geradas utilizando o protocolo de Diffie-Hellman, em vez de pares de chaves assimétricas pré-definidos.

2.2 Decisões tomadas

2.2.1 Ações definidas para o servidor

- **TalkToPlayer** : Envia uma mensagem para um jogador específico;
- **hello** : Mensagem de entrada no servidor, aqui são criadas as chaves de *Diffie Hellman*;
- **req_login** : Pedido de *login* de um utilizador, verficação se foi o primeiro a entrar (será host), caso não seja verifica se o jogo está cheio, caso não esteja adiciona o jogador e aguarda início pelo host;
- **start_game** : Envia uma mensagem para todos os jogadores a informar que o *host* iniciou a sessão;
- **ready_to_play** : Envia uma mensagem para todos os jogadores a informar que o *host* iniciou o jogo, após cifragem do baralho e distribuição de peças;
- **get_game_propreties** : Update com recolha das definições do jogo;
- **reg_points** : Escrita dos pontos;
- **KeyToPiece** : *Player* envia a chave que usou para cifrar a peça, servidor envia essa chave para o player que tirou a peça do stock;
- **selectionStage_end** : Fase de seleção terminou, envia para todos os jogadores a nova ação (Fase Commitment);

- **prep_stage_end** : Fase de preparação de-anomização (3.4.6) terminou. Guarda a lista de chaves públicas preenchida pelos jogadores e atualiza a nova ação (Fase de-anomização)

As seguintes ações são feitas após uma verificação de que é o jogador correto a jogar:

- **encryptDeck** : Recebe o baralho cifrado do jogador, atualiza o seu baralho e passa a vez para o próximo jogador
- **get_pieceInGame** : Um jogador tirou um peça do stock durante o joga e o servidor começa a pedir as chaves
- **get_piece** : Recebe a confirmação que a tradução da peça correu bem
- **tuploToPiece** : Recebe um pedido de tradução do tuplo e envia essa tradução para o jogador
- **revelationStage** : Recebe as chaves usadas para cifrar as peças que estão nas mãos dos jogadores e envias para todos os jogadores, passa a vez para o próximo jogador
- **bitCommit** : fase em que o servidor regista todos os dados revelados por parte dos clientes. no que toca ao calculo dos respetivos *bit commitments*;
- **verifyBC** : fase de re-cálculo do *bit commitment* do jogador vencedor, utilizando os dados por este revelados no momento em conjunto com os que revelou no início da partida. Também aqui é realizada a comparação das peças na mão inicial do jogador com as peças utilizadas ao longo da partida
- **prep_stage** : Guarda a lista de chaves públicas preenchida pelos jogadores;
- **de_anonymization_done** : Anonimização das peças feita, o jogo está pronto a ser jogado;
- **play_piece** : O jogador jogou uma peça e atualiza as peças que estão na mesa e passa a vez para proximo jogador
- **pass_play** : O jogador não jogou nenhuma vez e passa a vez para o próximo jogador
- **cheat_detected** : Envia uma mensagem para todos os jogadores a informar que houve batota e o jogador que fez a batota;
- **cheat_end_game** : Termina o jogo.

2.2.2 Ações definidas para o cliente

- **KeyToPiecePlayer** : Recebe do servidor a chave utilizada para cifrar a peça;
- **whatIsThisPiece** : Envia para o servidor a chave utilizada para cifra a peça ;
- **tuploToPiece** : Recebe do servidor o valor da peça correspondente a um tuplo;
- **login** : Faz o pedido de login na mesa e manda o valor de Diffie-Helman;

- **you_host** : Recebe uma mensagem a dizer que é o host da mesa ;
- **new_player** : Recebe informação que um novo jogador entrou na mesa;
- **waiting_for_host** : Espera que o host comece o jogo;
- **TalkToPlayer** : Falar com um jogador;
- **host_start_game** : O host começou o jogo;
- **cheat_detected** : Envia uma mensagem com o cliente que fez a batota e a pedir o término do jogo;
- **rcv_game_propreties** : Recebe informações sobre o estado do jogo;
- **end_game** : O jogo acabou e recebe quem ganhou o jogo;
- **reg_points** : fase em que o cliente pode inserir ou não Cartão de Cidadão e informa o servidor acerca dos seus pontos e a quem os atribuir;
- **wait** : Espera pela sua vez de jogar;
- **disconnect** : O servidor desligou-se e o cliente vai se desligar a seguir.

Ações do Cliente aquando a ação TalkToPlayer

- **OpenSession** : Estabelecer sessão com um jogador;
- **SessionEstabelicida** : Sessão estabelecida com um jogador;
- **get_piece** : tira um peça do stock e envia o stock para outro jogador ao caso;
- **prep_stage** : adiciona chave pública à lista de chaves públicas (ver mais secção 3.4.6)

Ações do Cliente aquando a ação rcv_game_propreties

- **de_anonymization_stage** : faz a tradução dos tuplos que têm na mão;
- **encryptDeck** : cifra o baralho e envia para o servidor;
- **get_piece** : tira um peça do stock e envia o stock para outro jogador ao caso;
- **bitCommit** : geração de 2 valores aleatórios e cálculo do bit commitment do jogador, seguido da sua divulgação;
- **revelationStage** : envia para o servidor as chaves que usou para cifrar as peças que estão nas mãos dos jogadores;
- **prep_stage** : adiciona chave pública à lista de chaves públicas (ver mais secção 3.4.6)
- **play** : joga um peça e envia-a para o servidor.

Capítulo 3

Funcionalidades implementadas

Como base do projeto, foi utilizado o código disponibilizado pelo colega Filipe Vale (link). Para além disso, os protocolos de segurança implementados foram reaproveitados do trabalho desenvolvido ao longo do semestre na componente prática da cadeira em questão.

3.1 Proteção das mensagens trocadas

As mensagens entre Player/Table Manager e entre Players são cifradas e decifradas com cifras simétricas Advanced Encrypion Standard (AES) no modo Cipher Block Chaining (CBC), sendo utilizado um Initialization Vector (IV) aleatório e um padding PKCS#7. Para cada sessão Player/Table Manager e entre Players existe uma cifra simétrica diferente. A chave de cada cifra simétrica é gerada através do processo Password-Based Encryption (PBE), onde a palavra-passe, que permite gerar a chave, foi combinada a partir do método de Diffie-Hellman.

3.2 Estabelecimento de Sessões entre jogadores e supervisor da mesa

Quando um jogador inicia uma sessão, envia uma mensagem de *hello* para o supervisor da mesa. Este faz um de pedido de *login* e retribui o seu valor de Diffie-Hellman. De seguida, o jogador envia o seu nome e o seu valor de Diffie-Hellman para o supervisor da mesa. Aqui, este vê se já existe um host na mesa e, caso não exista, responde ao jogador a atribuir-lhe o cargo de host. A partir desta mensagem as comunicações entre os dois já utiliza a cifra simétrica referida na secção 3.1.

3.3 Estabelecimento de Sessões entre jogadores

Quando um jogador novo entra na mesa, os outros jogadores enviam uma mensagem de *openSession* para ele, que contém o valor de Diffie-Hellman. O novo jogador responde enviando a seu valor Diffie-Hellman, e a partir daqui as mensagens entre os jogadores utiliza a cifra simétrica.
Para os jogadores falarem entre si, as mensagens primeiro têm que passar pelo supervisor da mesa. Este apenas consegue ver quem criou a mensagem e para quem é destinada. O seu

conteúdo está cifrado com cifra simétrica dos dois jogadores, logo, o supervisor da mesa não a consegue interpretar.

3.4 Protocolo de distribuição segura do baralho

A criação do baralho e a distribuição das peças pelos diferentes jogadores de uma forma segura é bastante complexa. Assim, como sugerido no enunciado do projeto, este protocolo foi dividido em diferentes fases:

3.4.1 Fase de Pseudonimização

Para a criação é usado o ficheiro *pieces*, onde estão os valores das peças do dominó. Para cada peça é atribuído um número aleatório e é gerada uma chave aleatória de tamanho de 5 símbolos, onde os símbolos são letras maiúsculas e algarismos. É verificado se esta chave foi usada noutra peça. Se sim, é gerada outra chave até que está seja única. Com isto é criado o pseudónimo da peça através do algoritmo utilizado no HMAC, onde a função de dispersão utilizada foi a SHA-256.

3.4.2 Fase de Randomization

Neste momento, o *Table Manager* faz circular o pseudo deck (N(i, Pi)) criado na fase anterior. Cada jogador cifra cada "peça"(tuplo (i, Pi)) e, antes de passar para o jogador diferente, baralha os tuplos cifrados. A cifra utilizada nesta fase é a mesma explicada na secção 3.1. É de realçar que cada tuplo é cifrado com uma *key* diferente. Internamente, cada jogador guarda num dicionário o resultado da cifra de cada tuplo e a chave usada o mesmo.

No ficheiro de teste *rand_stage.py* é possível verificar o que cada jogador faz nesta fase.

No final desta fase, ninguém sabe a que peça corresponde cada tuplo.

3.4.3 Fase de seleção

Depois do baralho ser cifrado e baralhado de modo a que ninguém, nem mesmo o *Table Manager*, saiba a que peça pertence cada tuplo, o baralho é novamente passado entre os jogadores, de modo secreto, onde estes podem escolher entre duas opções:

- Retirar peça

  De modo a dificultar o possível rastreamento e descoberta de uma peça que um jogador tirou, a probabilidade desta ação acontecer é de apenas 5%.

- Trocar peça ou não fazer nada

  Neste momento o jogador tem de realizar uma outra escolha, trocar uma peça, ou seja, retirar uma do baralho e devolver uma sua ao baralho, ou não efetuar mudanças. A probabilidade de cada uma das ações é 50%·

Uma diferença para a fase anterior, em que o baralho também era distribuído por todos os jogadores, é que agora este passa diretamente para outro jogador, não passando pelo *Table Manager*, como explicado na secção 3.3.

Esta fase termina quando todos os jogadores têm a sua mão completa.

3.4.4 Fase Commitment

Este processo utiliza 2 números inteiros aleatórios e o conjunto de peças numa função de hashing por chave (biblioteca *hmac* em Python). Esta função utiliza como chave a multiplicação dos números aleatórios referidos e, como mensagem, as peças do jogador, cifradas em Base64. Finalmente, para a produção do *digest*, foi utilizado o algoritmo SHA-256, através da biblioteca *hashlib*. Após ter o valor calculado (chamemo-lo BC), cada jogador torna público um tuplo contendo o primeiro número aleatório utilizado (R1), o valor de BC, e também o seu pseudónimo, de forma a ser armazenado por parte do Table Manager.

3.4.5 Fase de revelação

Nesta fase, o supervisor da mesa envia mensagens aos jogadores, pela ordem inversa que foi usada na fase 3.4.2, para estes retribuírem as chaves que usaram para cifrar as peças que não estão no stock. Para isso o supervisor envia o baralho todo para o jogador e este vê as peças que não estão no stock. Quando o supervisor recebe as chaves de um jogador, este decifra as peças correspondes que estão no baralho todo e também ele transmite as chaves para todos os jogadores que estão na mesa, para eles poderem também decifras as peças que têm na mão.

3.4.6 Fase de preparação de-anomização das peças

Nesta fase, a mão de cada jogador é do tipo [(0, "a"), (3, "b"), (5, "c")] onde o primeiro elemento de cada tuplo é o index (i) da peça e o segundo elemento é o pseudónimo (Pi).

Uma lista vai sendo passada diretamente de jogador para jogador, correndo todos, onde cada um vai preencher a posição dessa mesma lista na qual têm uma peça com esse index.

Os jogadores quando recebem esta lista escolhem entre duas opções:

- Preencher lista de chaves públicas

  Para preencher a lista é necessário primeiro gerar as chaves assimétricas. Neste sentido, para a geração das chaves assimétricas foi utilizado o algoritmo RSA, implementado da mesma forma que nas aulas práticas da cadeira. Isto está presente no ficheiro *security.py*, na classe *AsymCipher*, e num outro ficheiro de teste e simulação do que cada jogador faz nesta fase (*test_asym.py*). Como dito anteriormente, os jogadores preenchem a posição adequada da lista com a chave pública e guardam a chave privada no tuplo da peça.

  De modo a dificultar o rastreamento e a possível descoberta de quem adicionou a chave pública à lista, a probabilidade desta ação acontecer é de apenas 5%.

- Abster-se de fazer qualquer mudança

A lista vai sendo passada diretamente entre os jogadores até que todos tenham adicionado as chaves de todas as peças que têm na mão.

No final desta fase, a mão de cada jogador passa a ser do tipo [(0, "a", k-), (3, "b", k-), (5, "c", k-)] onde -k"corresponde à chave privada.

As posições da lista de chaves públicas que não estiverem preenchidas indicam os indexes das peças que ainda estão no baralho.

3.4.7 Fase de de-anomização das peças

Nesta fase, o supervisor da mesa percorre a lista de chaves e, para cada uma, gera um tuplo (k,peça), onde o k corresponde à chave utilizada para gerar o pseudónimo, depois cifra este com a chave corresponde da lista e adiciona o resulta da cifra numa nova lista na mesma posição que está a chave. No fim, envia a nova lista para todos os jogadores e aqui cada jogador decifra os seus tuplos e calcula os pseudónimos com os valores que recebeu para confirmar que estes valores são os corretos e depois substitui os tuplos que têm na mão pela peça que recebeu do supervisor da mesa.

3.4.8 Uso do baralho

Para um jogador poder utilizar o stock durante o jogo, ele precisa de dizer ao supervisor da mesa que foi buscar uma peça cifrada e que peça cifrada é. A partir daqui o supervisor da mesa vai percorrer os jogadores pela ordem inversa que foi utilizada na fase 3.4.2 para pedir as chaves que foram utilizadas e cada vez que este recebe uma chave ele transmite essa chave para o jogador que tirou a peça. O jogador que recebe a chave decifra e verifica se o resultado final é um tuplo. Caso seja, envia uma mensagem ao supervisor da mesa para este fazer a sua tradução. De seguida, o supervisor da mesa gera um tuplo (k,peça), onde k é a chave utilizada para gerar o pseudónimo e envia para o jogador. Quando este o recebe, verifica se os valores são os corretos e substitui a peça cifrada que tirou pela peça que recebeu.

3.5 Validação das peças jogadas por cada jogador

Para validação das peças jogadas no final do jogo, na sua totalidade, e assim confirmar a vitória de um dos jogadores, foi implementado o cálculo de um *bit commitment* para cada jogador.

3.5.1 Cálculo do bit commitment

Os valores de *bit commitment* são calculados como mencionado em 3.4.4, divulgando 2 elementos desse processo para a posterior verificação, abaixo mencionada.

```
MSG--> {'action': 'bitCommit', 'userData': (882, 'Q71QfC2rnYe6H/U6SoOkfY+xeqvKAm3DZQ36j0djRy0=', 'T66I')}

bitCommit
MSG--> {'action': 'bitCommit', 'userData': (857, '7A1X3EsAJCqFc8bQ2fdk39U8Uy+CRS5OCH4pVx2Oti4=', '3NI6')}

bitCommit
MSG--> {'action': 'bitCommit', 'userData': (160, '+3OanRRGFJNEMbdHNLmolQC0wwgJHB+JRDXk57O2Gkc=', 'LSWN')}
```

Figura 3.1: Divulgação dos BCs e valor inicial de cálculo

3.5.2 Verificação do conjunto de peças jogadas

No final do jogo, após se terem jogado todas as peças possíveis e determinado um vencedor, este apenas será assim proclamado após divulgar o segundo número aleatório utilizado e o conjunto de peças, ainda cifradas, obtidas no início (R2 e T). Juntando estes dados aos previamente divulgados, o Table Manager irá calcular de novo o valor de Bit Commitment (BC') e decifrar T, de forma a comparar com as peças prévias com as jogadas durante a partida. Caso BC seja diferente de BC' ou alguma peça de T não estiver incluída no conjunto de peças jogadas, assume-se que o vencedor não cumpriu as regras na totalidade.

3.6 Possibilidade de batota

Foram pensadas várias formas de fazer batota, entre elas:

- Jogar uma peça já jogada;
- Jogar múltiplas peças numa jogada;
- Jogar peças que estão na mão de outra pessoa;
- Repetir turnos;
- Passar um turno sem ir buscar;
- Jogar uma peça aleatória invés de ir buscar (quando não tem jogadas possíveis).

Entre estas, a forma de batota escolhida foi a última. Sendo assim, quando um jogador não tem jogadas possíveis para as peças que tem na mão, ao invés de ir buscar uma peça e passar o turno, este irá jogar uma peça aleatória da sua mão.

Há uma possibilidade de 1 em 200 de o jogador fazer batota no momento em que tem que ir buscar uma peça. Um jogada inválida (batota) irá seguir os mesmo procedimentos que uma normal e o *score* do jogador irá aumentar em cinco pontos como se tivesse feito uma jogada normal.

3.7 Protesto contra batota

3.7.1 Deteção de batota

De forma a detetar se foi feita batota, todos os jogadores no início do seu turno procedem a uma verificação da peça jogada no turno anterior. Para isto são usadas as peças dos extremos da mesa, e estas são de seguida comparadas com as peças anteriores a cada uma. Se os valores da peça do extremo e da anterior ao extremo não forem iguais, conclui-se que um jogador fez batota. Sabendo que esta verificação é feita todos os turnos, em caso de batota sabe-se sempre que quem fez a batota foi o jogador anterior.

3.7.2 Protesto

Caso seja detetada batota através do método explicado na subsecção anterior 3.7.1, o jogador que a deteta faz um protesto de imediato. Este protesto consiste numa mensagem enviada para o servidor a informar qual foi o jogador que fez a batota (jogador que jogou

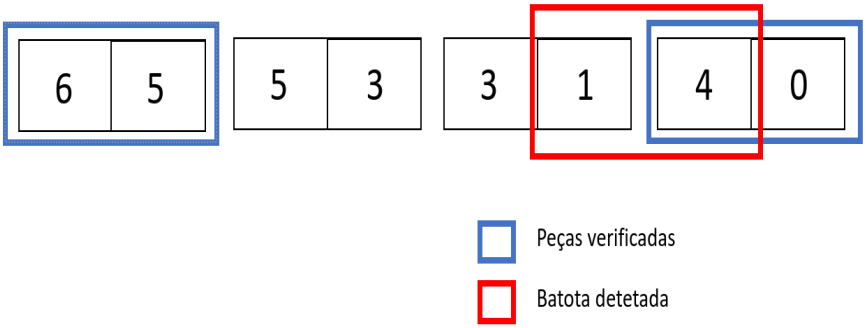

Figura 3.2: Deteção de batota

previamente). De seguida, o servidor envia uma mensagem para todos os jogadores a informar a existência de irregularidades e de quem as praticou. Após o envio da mensagem o jogo é terminado pelo servidor.

3.8 Game accounting

Cada jogador recebe cinco pontos por cada peça jogada, estes pontos são instantaneamente somados ao *score* total do jogador. Quando um jogador é declarado vencedor este recebe 100 pontos. Se um jogo chegou ao fim e foi declarado um vencedor é assumido que todos os jogadores estão de acordo com as pontuações e não foi detetada nenhuma batota, visto que não foi feito qualquer protesto, sendo assim possível fazer-se a reivindicação dos pontos.

3.9 Reivindicação dos pontos por uma entidade através do Cartão de Cidadão

De forma a uma entidade reivindicar os pontos de uma partida, deve submeter o seu Cartão de Cidadão para leitura, de forma a que seja efetuado um registo de pontos por pessoa, sendo possível o aumento de pontos caso a mesma entidade reivindique os pontos associados a outros pseudónimos, noutras partidas. Este registo tem por base a leitura do *Serial Number* do Certificado de Autenticação do Cartão de Cidadão, e a sua cifragem, também baseada em funções de hashing com chave, para que seja impossível interpretar/utilizar estes identificadores no caso de o ficheiro de registo de pontos ser comprometido. A chave utilizada na cifragem depende do *Serial* do Cartão, pelo que o resultado será sempre idêntico, o que torna possível obter uma representação inequívoca de uma entidade e a acumulação progressiva de pontos ao logo de várias partidas. Caso esta reivindicação não seja realizada, os pontos serão registados como pertencentes ao pseudónimo que os angariou ao longo de uma partida.

3.10 Apanhar peças durante um jogo empatado

Quando um jogador não tem uma jogada válida a realizar, este retira uma peça do baralho. Se, com a peça retirada do baralho, ainda não for possível realizar uma jogada válida, retira

novamente outra peça. Este ciclo mantém-se até que não haja mais peças no baralho e o jogador passa a vez para um outro jogador. Como explicado anteriormente, é nesta fase que também pode fazer batota.

Capítulo 4

Dificuldades

4.1 Interpretação

Interpretação do enunciado especialmente na parte do protocolo de distribuição segura do baralho.

4.2 Conceber o esquema de comunicação entre servidor e cliente

O esquema composto por um servidor e clientes é por várias vezes estudado. Contudo, em poucas ocasiões este é realmente implementado pelos alunos, tendo apenas que o utilizar e implementar novas funcionalidades. Assim, revelou-se também um momento de aprendizagem para os elementos do grupo, implicando, naturalmente, mais esforço e tempo dedicado ao projeto.

Capítulo 5

Conclusões

Da realização deste projeto é possível retirar que a elaboração de um sistema seguro, capaz de simular um jogo de tabuleiro, ainda que simples, é de elevada complexidade. Ainda assim, o sistema pode não ser 100% seguro, e sistemas de elevado poder computacional, por exemplo, podem decifrar elementos do jogo antes da sua utilização. É também importante salientar que este sistema é executado apenas numa máquina, tendo o grupo perfeita noção que uma execução distribuída entre várias máquinas (simulando servidor e clientes diferentes) poderia dar aso a ainda mais vulnerabilidades. Possível trabalho futuro passaria por implementar a mencionada distribuição entre máquinas, interface gráfica e, sobretudo, diferentes e mais complexos algoritmos criptográficos.

Bibliografia

[1] Conteúdos lecionados durante o período letivo: `http://sweet.ua.pt/andre.zuquete/Aulas/Seguranca/20-21/index.html`

[2] Mecânicas do jogo e formato de cliente/servidor baseados no modelo do colega Filipe Vale: `https://github.com/FilipeMiguelVale/Secure-Domino`

[3] Questões relacionadas com a linguagem de programação Python e dúvidas de esclarecimento rápido `https://stackoverflow.com`