Secure, multi-player online domino game

Security

(89162) jacinto.lufilakio@ua.pt
(88747) marcia.pires@ua.pt
(89077) renatovalente5@ua.pt
(89286) tomasfilipe7@ua.pt

Resumo

Este relatório tem como objetivo descrever os passos seguidos para o desenvolvimento de um jogo de dominó seguro com as seguintes funcionalidades:

- Trocas de mensagens protegidas (criptografia, autenticação, etc.);
- Estabelecer sessões entre o jogador-jogador e entre jogador- *Table Manager*;
- Distribuição segura do baralho;
- Validação das peças jogadas pelos jogadores;
- Protesto contra batota;
- Possibilidade de fazer batota (jogador batoteiro);
- Contar os pontos do jogo;
- Reivindicação de pontos para uma identidade fornecida por um Cartão de Cidadão;
- Seleção de peças do estoque durante o jogo (quando as peças em mão não podem ser jogadas).

Conteúdo

Capítulo 1

Introdução

Este relatório tem como objetivo detalhar os passos efetuados para o desenvolvimento do jogo de dominó com a devida segurança. O jogo é composto por um **Table Manager** (server.py) e um **conjunto de jogadores** (client.py). Cada jogador necessita de autenticação para poder entrar na partida. Após a validação da autenticação, os jogadores possuem um pseudónimo aleatório utilizado apenas durante a partida, ao qual está associado o seu Serial Number, retirado do Citizen Card (CC). No final de cada partida se todos os jogadores concordarem com o resultado, o jogador vencedor receberá 5 pontos que serão acumulados aos pontos anteriores deste mesmo jogador. Para isso, usamos o pseudónimo do jogador para saber o Serial Number associado no início do jogo e tranferir assim os pontos para o CC do vencedor. A responsabilidade de criar o baralho com peças de dominó e o seu embaralhamento inicial é de inteira responsabilidade do gerente da mesa.

Cada jogador faz um compromisso da sua mão inicial com a mesa antes do jogo começar, isto não revela o seu jogo à mesa. Os jogadores não podem jogar peças que não têm na sua posse e, portanto, caso seja detetada batota por parte de um jogador é preciso recorrer à verificação do compromisso para se saber que jogador fez batota. Tanto o gerente de mesa como os jogadores podem reclamar de batota. A batota pode ser feita ao jogar peças repetidas ou ao jogar peças que os jogadores não têm. Os jogadores podem detetar e reclamar se outro jogador está a usar uma das suas peças.

Capítulo 2

Funcionalidades do Jogo

Este programa consiste num jogo de dominó simples, que segue as seguintes regras:

1. Cada jogador começa por retirar **5 peças do baralho** para a sua mão.
2. O jogador **Host** (primeiro a ligar-se ao servidor) coloca a peça que quiser na mesa.
3. Cada jogador à vez, irá **colocar uma peça** cujo número de um dos lados seja igual a um número de um dos lados que estão nas margens da mesa.
4. O jogo **termina** no momento em que um jogador fique sem peças na mão ou o baralho chegar ao fim.
5. O jogo pode ainda ser abortado se um jogador for apanhado a cometer uma **fraude** sem possibilidade de reverter a jogada.
6. No final do jogo é possível concordar com o resultado e distribuição dos pontos. Se todos concordarem os pontos serão adicionados à conta autenticada pelo CC do jogador.

Capítulo 3

Sistema de Cifras e algoritomos criptográficos

Todos os sistemas de cifras e algoritmos criptográficos usados têm como base as bibliotecas usadas nas aulas (*cryptography hazmat*)

No nosso projeto, tanto para cifrar as mensagens das sessões entre jogadores e *Table Manager* quanto para cifrar as peças, foi usada a **cifra de blocos Advanced Encryption Standard (AES)** no modo **CBC**. Para a troca de mensagens foi inicialmente implementada a cifra **Advanced Encryption Standard Galois/Counter Mode (AES-GCM)** dada a rapidez que esta traria para esta troca, no entanto a sua implementação estava a gerar alguns erros que não fomos capazes de resolver e portanto optámos por escolher o modo **Cipher Block Chaining (CBC)** que também nos oferece um elevado grau de confidencialidade. Para ambas as situações optámos por **cifras simétricas**, pois estas dependem de menos poder computacional comparativamente às cifras assimétricas, logo são **mais rápidas** e assim não diminuem a fluxo do jogo. Tendo isso em mente usamos chaves de 256 bits, pois estas aumentam a segurança da cifra, tornando-a mais difícil de ser atacada por força bruta. Em ambos os casos decidimos também usar este tipo de cifra pois, na cifra das peças, a chave terá que ser posteriormente revelada, e na cifra das mensagens a chave simétrica é criada a partir das chaves públicas entre os pares pelo método de Diffie-Hellman.

Para o estabelecimento de chaves de sessão foi então utilizado o algoritmo de **Diffie-Hellman**. Este, apesar de ser considerado um algoritmo de cifra assimétrica, é na verdade um **algoritmo de distribuição de chaves (de sessão)**. Este algoritmo baseia a sua segurança na **dificuldade de cálculo de logaritmos modulares de grandes números** e através dele somos capazes de estabelecer um segredo compartilhado entre duas partes, que pode ser usado para uma comunicação secreta, para a troca de dados numa rede pública. Foi escolhido este algoritmo em vez do Diffie-Hellman de curva elíptica, dado ser mais fácil a sua implementação.

Na **autenticação** dos jogadores com o cartão, quando guardamos o identificador único do cartão, decidimos usar **Triple Data Encryption Algorithm (Triple DES)** pois este é menos custoso para o hardware e como é o identificador do cartão é pessoal e intransmissível a cifra usada pode ser considerado um capricho.

Para guardar o Serial Number aquando o login do utilizador na nossa base de dados é utilizada uma encriptação de **Triple DES** no modo **CBC**, este usa 3 chaves de 64 bits, portanto no total usa um comprimento de chave de 192 bits. A primeira parte do processo é uma criptografia Data Encryption Algorithm (DES) regular, a segunda parte é uma decifragem DES e, por fim, usa uma criptografia DES novamente. Usando assim 3 chaves diferentes. Triple DES é considerado mais fácil de implementar em hardware e software do que AES e está presente na maioria dos sistemas, bibliotecas e protocolos.

Usamos também a **cifra assimétrica Rivest-Shamir-Adleman (RSA)**, pois o *Table Manager* tem que circular um *array* com as peças de cada jogador por todos os jogadores, e para prevenir que os jogadores conheçam as peças de outros jogadores, o *Table Manager* cifra com a chave pública desse jogador, permitindo assim que apenas esse jogador possa saber a peça que lhe destina. Apesar de ser uma cifra comparativamente lenta, como esta é uma fase crucial para o jogo (estritamente necessário manter a confidencialidade das peças de cada jogador) mesmo os outros jogadores sabendo a chave pública dos outros, é praticamente impossível de encontrar a chave através de força bruta, e mesmo que fosse possível não o seria durante o tempo em que o jogo decorre.

São usadas funções *hash Secure Hash Algorithm 2-bit (SHA-2)* para a geração tanto das chaves usadas nas cifras das peças (*Secure Hash Algorithm 512-bit (SHA-512)* juntamente com *key derivation*), como para gerar o *bit-commitment* das mãos cifradas dos jogadores (*SHA-512*). Escolhemos estas funções em vez de outras como, *Message-Digest Algorithm 5 (MD5)* e *Secure Hash Algorithm 1-bit (SHA-1)* pois nestas já foram detetados problemas de colisões nas suas *digests*.

Capítulo 4

Detalhes de Implementação

4.1 Servidor

O servidor, ou *Table Manager*, irá expôr um **endpoint** de ligação por onde os clientes irão trocar as suas mensagens. Este é responsável por gerir todo o processo do jogo, desde formar as tabelas de anonimato dos jogadores, criar o baralho de peças e distribuí-las corretamente, **controlar** o fluxo do jogo, receber denúncias dos jogadores, apanhar jogadores a fazer batota e associar as identidades do jogo.

Assim que se ligarem ao servidor, os clientes terão de se apresentar e provar o domínio do seu **pseudónimo**, assim como uma ligação entre o pseudónimo e a sua identidade.

Quando o número máximo de clientes for atingido, o servidor irá iniciar o jogo e distribuirá 5 peças a cada jogador, esperando pelo cliente anfitrião que decida quando quer começar o jogo.

4.2 Cliente

O cliente, ou **jogador**, é uma entidade que interage com o utilizador de modo a permitir que este participe num jogo. Este irá tentar conectar-se ao servidor com uma autenticação e caso seja validada, fará *login* no *server*. O primeiro cliente a fazer *login* é considerado o jogador anfitrião.

A cada jogador é atribuido um pseudónimo aleatório que é apenas utilizado durante um jogo.

Este pseudónimo estará ligado às ações do jogador, como o compromisso da mão do mesmo, interagir com as peças, protestar sobre uma fraude cometida e concordar ou discordar com a pontuação final.

4.3 Autenticação

Para a **validação da autenticação** do jogador no jogo é preciso que o leitor de cartões e o cartão estejam ligados ao computador. Foi então implementada uma comunicação de **Desafio-Resposta** com cartão para a sua validação. Na fase de autenticação o servidor gera um **desafio random** e envia para o cliente. Este assina o desafio com a sua chave privada retirada do CC e devolve o resultado ao servidor que irá verificar a assinatura com a chave pública conhecida do CC. Caso a verificação seja válida, a autenticação é também considerada válida,; caso contrário o cliente é desligado e terá de tentar autenticar-se de novo. Este código pode também ser consultado no ficheiro *"authentication.py"*.

4.4 *Serial Number*

O ***Serial Number*** (número de série do cartão) é utilizado para a associação do cliente com o pseudónimo criado para o jogo em específico. Após uma autenticação bem sucedida por parte do cliente, o servidor irá atribuir um pseudónimo ao cliente com a associação do seu *Serial Number* encriptado com Triple DES. Esta associação será guardada durante o jogo num dicionário no servidor para que, no final do jogo, seja possivel a atribuição dos pontos, respetivos ao cliente cujo pseudónimo é o vencedor. Estes pontos

serão guardados e acumulados num ficheiro Comma-Separated Values (CSV) (*"data.csv"*) que conterão os pontos de todos os jogadores vencedores e o seu respetivo *Serial Number* encriptado.

4.5 Base de dados

Os pontos dos jogadores são guardados num ficheiro **CSV**, onde estão associados os pontos dos jogadores vencedores com o respetivo *Serial Number* do CC encriptado.

4.6 Estabelecimento de chaves de sessão

Após ser efetuada a validação do jogador através do seu CC, é iniciado o processo de estabelecimento de **chaves de sessão**. Para isso, tal como referido na secção *Sistema de Cifras e de algoritmos criptográficos*, foi utilizado o algoritmo de **Diffie-Hellman**. Assim, tanto o *Table Manager* quanto os jogadores geram cada um, um **par de chaves**: uma privada, que se mantém privada o jogo todo, e uma pública, que é partilhada entre os pares. Através da chave pública, é feito o cálculo da chave de sessão. Além do esperado cálculo desta chave, é também feita uma derivação da mesma utilizando **HKDF**, uma função de derivação de chaves, que utiliza o algoritmo de *hash SHA-256*. Esta derivação permite diminuir a probabilidade de ataques de força bruta.

Primeiro são estabelecidas as chaves de sessão entre **jogador - *Table Manager***. Em seguida, são criadas as sessões Transmission Control Protocol (TCP) entre **jogador - jogador**, em que a troca de chaves públicas é cifrada utilizando a chave de sessão estabelecida com o *Table Manager* para evitar que terceiros possam interceptar a chave publica destinada ao par da sessão, isto, assumindo que o *table manager* é de confiança. As mensagens são ainda assinadas com a mesma chave de sessão, utilizando o código de autenticação de mensagens **Hash-based Message Authentication Code (HMAC)**, que por sua vez envolve novamente a função de *hash Secure Hash Algorithm 256-bit (SHA-256)*.

No final de todas as chaves de sessão serem trocadas, podemos passar para os próximos estágios antes do início do jogo.

4.7 Troca de mensagens

Para que seja realizada a troca de mensagens, tanto entre jogador e *Table Manager* quanto entre um par de jogadores, é estabelecida uma sessão TCP, com chaves simétricas associadas, sendo que a comunicação entre os jogadores é encaminhada através do *Table Manager*.

Estando as sessões estabelecidas, a comunicação entre jogadores necessita ser **confidencial** e portanto, a partir desse momento, todas as partes das mensagens cujo conteúdo partilhado seja considerado crítico, são cifradas. Esta cifragem foi realizada utilizando a cifra **AES** com o modo **CBC** tal como referido na secção *Sistema de Cifras e algoritmos criptográficos*. Como chave simétrica foi utilizada a chave de sessão calculada anteriormente.

Além da confidencialidade, para garantir que as mensagens são **autenticadas** e para permitir o controlo de **integridade**, estas são assinadas utilizando o código de autenticação de mensagens **HMAC**, tal como referido anteriomente na secção *Estabelecimento de chaves de sessão*. Para esta autenticação são então usadas as chaves de sessão e as mensagens já cifradas, ou seja é utilizada uma abordagem **Encrypt-then-Message Authentication Code (MAC)** visto ser conhecida como a mais segura.

4.8 Batotas dos jogadores

4.8.1 Diferentes clientes

Tal como solicitado no enunciado, foi implementada a funcionalidade dos jogadores poderem fazerem batota. Esta batota consiste em utilizar peças que não estariam originalmente na mão do jogador. Para tal existem 2 clientes diferentes para a nossa aplicação:

1. **client.py -** Este cliente não irá tentar fazer batota

2. **client_cheating.py -** Este cliente irá tentar enganar tanto o *Table Manager*, como os outros jogadores.

4.8.2 Compromisso dos jogadores

No inicio da execução do programa, todos os jogadores terão de **comprometer** a sua mão através de um processo de *bit commitment*, de maneira a assegurar que os jogadores **não joguem peças diferentes** das peças da sua própria mão inicial, **não revelando** que peças é que cada jogador tem na sua mão.

Este processo é calculado através de uma *digest hash function* e **2 valores aleatórios**(R1 e R2).

4.8.3 Batota I - Jogar uma peça do deck ou que já está em cima da mesa.

A batota pode ser identificada pelo *Table Manager*, ou por outro jogador.

No primeiro caso, o *Table Manager*, dado que tem a informação de quais peças estão nas **mãos dos jogadores**, na **mesa** e consequentemente, também as que estão no **baralho**, analisa todas as peças que são colocadas na mesa e verifica se algum jogador está a jogar uma peça que deveria estar no baralho ou na mesa de jogo. Se for identificado algum tipo de fraude através deste método, a jogada é **invalidada** e é requisitado ao utilizador que **volte a jogar** uma peça.

4.8.4 Batota II - Jogar uma peça de outro jogador.

No segundo caso, visto que o *Table Manager* não sabe que peças estão na mão de cada jogador, fica ao cargo dos mesmos **denunciarem** alguém que tente utilizar uma das suas peças. Caso tal aconteça, este irá denunciar o tal jogador ao *Table Manager*, quer por sua vez irá utilizar os valores de **R1 e R2** (valores utilizados na criação do *bit commit*, juntamente com uma hash) do jogador batoteiro, de modo a repetir o processo de *bit commitment* e comparar com o *bit commitment* calculado no **inicio do jogo** de modo a verificar se este adulterou ou não a sua mão inicial. Se nada foi alterado,

então o *Table Manager* irá **desencriptar** a mão do jogador em análise e verificar se esta contém a peça jogada pelo mesmo.

De seguida, se a peça não for encontrada (O que significaria que a peça não estava na mão inicial do jogador), o *Table Manager* ainda irá recorrer a um **mapa** que este contém, onde foi guardado as peças que cada jogador tirou do *deck* durante o jogo na fase de compra. Se não encontrarmos uma relação entre a peça em causa e o jogador acusado de batota, o *Table Manager* confirma então que o utilizador fez batota e aborta o jogo finalizando o mesmo como um empate.

Capítulo 5

Execução do Jogo

5.1 Configuração do Jogo

Para a execução do jogo, é preciso ter instalado o Python e as seguintes bibliotecas:

```
$ sudo apt update
$ sudo apt install python3-pip
$ sudo pip3 install pykcs11
$ sudo pip3 install cryptography
$ sudo pip3 install pandas
```

Tem também de ser instalado o *middleware* necessário para usar o CC, sendo distribuído em conjunto com um aplicativo de suporte que pode ser encontrado aqui.

5.2 Manual de Jogo

Após a instalação dos módulos necessários é possivel dar-se início ao jogo.

Para a execução do servidor, que terá de ser iniciado em primeiro, basta estar no diretório do projeto e executar:

```
$ python3 server.py
```

Para a execução do cliente (terá de ser executado em vários terminais para criar vários jogadores), basta estar no diretório do projeto e executar:

```
$ python3 cliente.py
```

Foi ainda criado um script que facilita a execução do jogo e acrescenta a possibilidade de analisar os pontos do cliente. Para isso basta executar o script no terminal e de seguida usar os atalhos consoante o pretendido:

```
$ source script.sh
```

Por cada cliente criado irá ser feita a validação da autenticação deste com um desafio-resposta, como estamos a usar o CC o cliente tem de inserir o Personal Identification Number (PIN) de Autenticação solicitado para ele mesmo aceder à sua chave privada. Após o servidor validar a autenticação do cliente, este é adicionado ao jogo, quando a partida estiver cheia, dar-se-á início ao jogo. No final do jogo cada cliente tem de concordar com o resultado final e, caso todos aprovem o resultado, serão adicionados 5 pontos ao vencedor; caso não concordem, não haverá venceder e ninguém ganha pontos. Em caso de empate também ninguém ganha pontos (por exemplo quando não há peças no baralho e os jogadores ainda têm peças na mão, sendo que nenhuma delas dá para jogar).

Capítulo 6

Resultados

Obtivémos um resultado bastante positivo face aos objetivos estabelecidos, pois implementámos todas as funcionalidades pedidas pelo enunciado, com alguns problemas que não conseguimos ultrapassar.

Ao usarmos apenas um CC, os pontos do jogador vencedor são guardados na base de dados sempre associados ao mesmo *Serial Number* cifrado.

É necessário esperar que o leitor de cartões faça a leitura completa do cartão antes de tentar iniciar uma sessão com outro cliente, senão existe a possibilidade de a leitura do cartão dar erro.

Neste jogo é possivel detetarmos diferentes tipos de batota por parte dos jogadores, permitindo assim um resultado justo entre todos os utilizadores.

Apesar do nosso esforço para contornar os erros ao longo do projeto, esporadicamente o programa é abortado numa fase inicial devido a uma excepção cuja causa não nos foi possível identificar. Também a meio do programa pode ser encontrada outra excepção relacionada à biblioteca que serializa as nossas mensagens para o formato *JavaScript Object Notation (JSON)*: este erro acontece quando a mensagem serializada é corrompida, não sendo possível desserializar a mesma.

Capítulo 7

Conclusão

Neste relatório foi descrita nossa solução de implementação por forma a obtermos um jogo com comunicação de mensagens seguras tanto entre *Table Manager* e jogadores, como entre pares de jogadores. O nosso principal objetivo ao desenvolver este projeto foi sempre tornar o nosso jogo o mais seguro e robusto possível, aplicando da melhor forma todos os conhecimentos que foram adquiridos na unidade curricular de Segurança, ao longo do semestre. Também nos fomos deparando com algumas limitações, quer fosse a nível de segurança através dos sistemas de cifra que fomos implementando, a nível de poder computacional ou mesmo a nível de complexidade de implementação. Contudo, procurámos ultrapassar todos os obstáculos que fomos encontrando, concluindo que o resultado final do projeto tem uma balanço bastante positivo.

Um agradecimento ao aluno Filipe Vale por nos forcener o código base para o jogo deste projeto.

Capítulo 8

Acrónimos

CC Citizen Card
CSV Comma-Separated Values
DES Data Encryption Algorithm
Triple DES Triple Data Encryption Algorithm
CBC Cipher Block Chaining
AES Advanced Encryption Standard
AES-GCM Advanced Encryption Standard Galois/Counter Mode
HMAC Hash-based Message Authentication Code
MAC Message Authentication Code
TCP Transmission Control Protocol
RSA Rivest-Shamir-Adleman
MD5 Message-Digest Algorithm 5
PIN Personal Identification Number
JSON JavaScript Object Notation
SHA-1 Secure Hash Algorithm 1-bit
SHA-2 Secure Hash Algorithm 2-bit
SHA-256 Secure Hash Algorithm 256-bit
SHA-512 Secure Hash Algorithm 512-bit

Capítulo 9

Bibliografia

- Código base do jogo de dominó - Filipe Vale
- André Zúquete, 2018, *Segurança em Redes Informáticas*, 5º edição, FCA.
- Middleware para usar o CC - https://www.autenticacao.gov.pt/cc-aplicacao
- https://cryptography.io/en/3.2/hazmat/primitives/asymmetric/rsa/
- https://cryptography.io/en/latest/hazmat/primitives/asymmetric/dh.html
- https://cryptography.io/en/latest/hazmat/primitives/cryptographic-hashes.html
- https://cryptography.io/en/latest/hazmat/primitives/key-derivation-functions.html
- https://docs.python.org/3/library/base64.html
- https://docs.python.org/3/library/pickle.html
- https://cryptography.io/en/2.7/hazmat/primitives/mac/hmac
- https://cryptography.io/en/2.8/hazmat/primitives/key-derivation-functions/
- https://cryptography.io/en/2.8/hazmat/primitives/asymmetric/dh/