

Segurança 2020-2021
Secure multi-player online Domino game
MIECT

Daniel Magueta, 68698

Diana Silva, 80239

João Abrantes, 79987

José Brás, 74029

Conteúdos

Introdução	3
Descrição do jogo	4
Funcionalidades	5
2.1. Privacidade das Mensagens	5
2.2. Autenticação do Cliente	6
2.2.1. Protocolo de Autenticação Sem Cartão de Cidadão	6
2.2.2. Protocolo de Autenticação Com Cartão de Cidadão	7
2.3. Pseudonimização da pilha de peças.	8
2.4. Protocolo de Baralhar (Randomization Stage)	8
2.5. Protocolo de Seleção (Selection Stage)	9
2.6. Protocolo de Comprometimento(Bit Commitment)	10
2.7. Revelation Stage	10
2.8. Tile de-anonymization Preparation Stage	11
2.9 Tile de-anonymization Stage	11
2.9. Stock Use	11
2.10. Contabilidade correta	11
2.11. Batota	12
Implementação	12
3.1. Problemas conhecidos e limitações	12
Conclusão	12

Introdução

Foi proposto, no âmbito da unidade curricular de Segurança, a criação e desenvolvimento de um jogo de Dominó, com o objetivo de manter o jogo em rede e de forma segura, que é a principal componente desta unidade curricular, a segurança.

O projeto contém o cliente (client.py), o Servidor(server.py), a Segurança(security.py) e o Cartão de Cidadão(C_Card.py), todos estes módulos trabalham em conjunto para produzir um jogo em rede e seguro.

Este jogo de Dominó é todo implementado com a ajuda de protocolos de segurança e autenticação da parte dos jogadores, que era o objetivo deste projeto, sendo este desenvolvido em Python. Estes protocolos e autenticação serão esclarecidos ao longo deste relatório.

Descrição do jogo

O jogo Dominó comum tem como regras que tem de ter um mínimo de dois jogadores, e então é atribuído, normalmente, 5 peças a cada jogador, sendo que o primeiro jogador a jogar é o que tiver a dupla mais alta. Cada jogador pode jogar para cima, para baixo, para a direita e para a esquerda de acordo com o que for favorável no momento, se um jogador não tiver peças para jogar vai a pilha buscar mais, se não existirem mais peças em pilha o jogador passa a sua vez ao próximo.

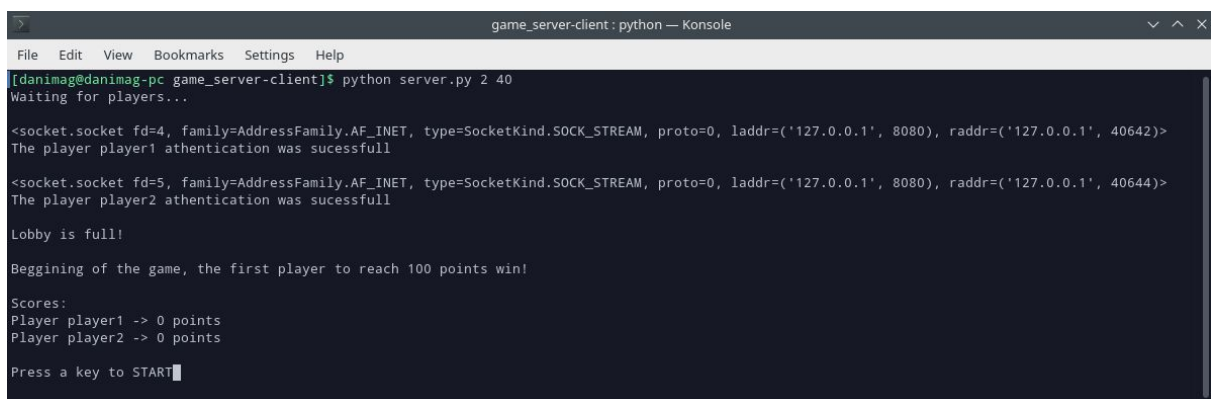
Neste projeto, o jogo Dominó funciona com um mínimo de 2 jogadores e um máximo de 4.

O servidor recebe quantos jogadores vão jogar, e mal receba que todos os jogadores se autenticaram começa o jogo. O servidor serve 5 peças a cada jogador ficando com as restantes em pilha.

Cada jogador pode jogar para a esquerda ou para a direita, e se não tiver peças para jogar tem de ir buscar a pilha até conseguir uma peça que possa jogar, se entretanto acabarem as peças da pilha e o jogador não tiver encontrado uma peça que possa jogar, tem de passar a sua vez ao próximo jogador. Ganha cada ronda quem acabar primeiro com as suas peças excepto se todos os jogadores tiverem passado a sua vez por não terem peças que dêem para jogar, a ronda termina e neste caso o jogador que tem menos pontos ganha o total dos pontos na mão dos outros jogadores menos os que ficaram na sua, o vencedor do jogo é o jogador que chega primeiro aos 100 pontos.

O jogo é iniciado correndo no terminal “python server.py <numplayers> <numtiles>”.

O parâmetro numplayers pode ser um número entre 2 e 4 que serve para inicializar o número de jogadores esperados e o parâmetro numtiles serve para escolher o número de peças com que se vai jogar. Ambos os parâmetros podem não ser inicializados pela consola, e nesse caso o número de jogadores será 4 e o número de peças 40.



```
game_server-client: python — Konsole
File Edit View Bookmarks Settings Help
[danimag@danimag-pc game_server-client]$ python server.py 2 40
Waiting for players...

<socket.socket fd=4, family=AddressFamily.AF_INET, type=SocketKind.SOCK_STREAM, proto=0, laddr=('127.0.0.1', 8080), raddr=('127.0.0.1', 40642)>
The player player1 authentication was successfull

<socket.socket fd=5, family=AddressFamily.AF_INET, type=SocketKind.SOCK_STREAM, proto=0, laddr=('127.0.0.1', 8080), raddr=('127.0.0.1', 40644)>
The player player2 authentication was successfull

Lobby is full!

Beggining of the game, the first player to reach 100 points win!

Scores:
Player player1 -> 0 points
Player player2 -> 0 points

Press a key to START
```

Funcionalidades

2.1. Privacidade das Mensagens

A privacidade das mensagens trocadas, numa fase inicial, é feita com criptografia assimétrica. A criptografia assimétrica utiliza um par de chaves sendo uma delas privada e outra pública, na qual a chave privada não é transmissível, isto faz com que haja confidencialidade sem existir troca prévia de mensagens, isto é, o originador da mensagem pode cifrar uma qualquer mensagem com o uso da chave pública do receptor, sendo que essa mensagem apenas pode ser decifrada utilizando a chave privada que o receptor possui. Foi utilizado este tipo de mensagens apenas para o objetivo de o cliente partilhar uma chave de sessão simétrica com o Servidor de uma forma segura. O algoritmo que foi utilizado para realizar esta criptografia assimétrica foi o RSA.

O algoritmo RSA foi utilizado não só devido ao facto de ter uma implementação mais simples, mas também porque pode ser usado para as assinaturas digitais que neste projeto iriam ser necessárias, tanto para a autenticação e reclamação dos pontos no final como para a integridade das mensagens.

Neste tipo de algoritmo podemos também definir o tamanho da chave utilizada, sendo que chaves maiores são mais seguras.

Neste projeto temos o par de chaves do servidor e do cliente (sendo utilizada a do cartão único ou, quando não é usado cartão, um gerador de par de chaves).

“PRINT DE GERAÇÃO DE PARES DE KEYS”

“getPublicKey()”

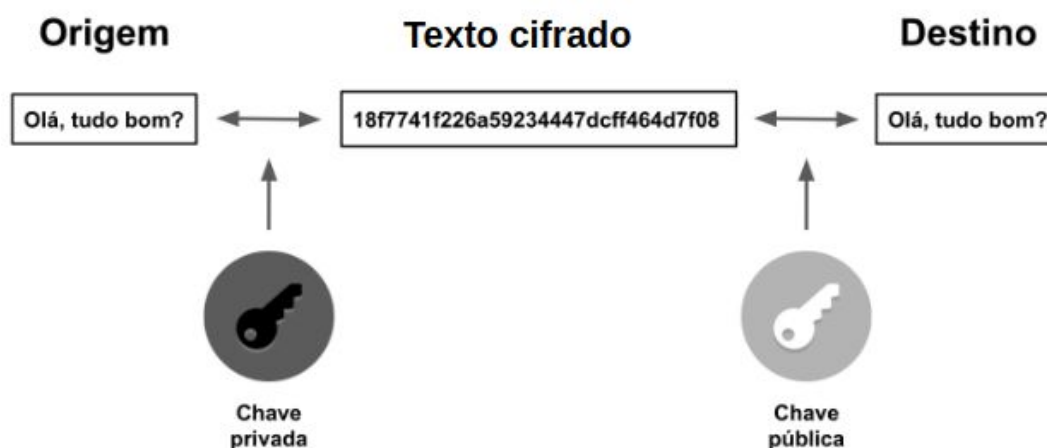


Fig.1: Criptografia Assimétrica

Como foi referido, após a troca da chave de sessão (através da cifragem com a chave pública do Servidor), foi sempre utilizado o algoritmo de cifra simétrica AES(128) quando fosse necessário garantir confidencialidade de uma mensagem entre um Cliente e um Servidor.

Apesar de à primeira vista a partilha da chave de sessão pelo meio de comunicação ser uma falha de segurança, é importante lembrar que esta chave de sessão só é decifrada com a chave privada do Servidor, uma chave que este nunca partilha. Além disso, a criptografia simétrica é computacionalmente menos intensiva, sendo um fator fundamental

neste contexto, visto que depois da fase de autenticação irão ser trocadas mensagens(seguras) frequentemente.

Foi utilizado o AES-128 pois, de todas as outras alternativas de implementação de criptografia simétrica(RC4, DES, etc) e com o auxílio de Fernet, que se encontra incluído na biblioteca cryptography do Python, acaba por ser a alternativa de mais simples implementação. Como Fernet apenas faz uso de AES-128, acaba por ser o utilizado.



Fig.2: Criptografia Simétrica

2.2. Autenticação do Cliente

2.2.1. Protocolo de Autenticação Sem Cartão de Cidadão

Para a autenticação sem Cartão de Cidadão, foi criado um protocolo de autenticação para proceder a autenticação do cliente/jogador no qual é enviado as seguintes mensagens:

- O cliente envia uma primeira mensagem que envia cifrada com a chave pública do servidor um conjunto de informação sobre ele próprio(tal como, pseudônimo, tipo de mensagem('AUTH0'), valor aleatório gerado para o servidor assinar, chave de sessão e uma chave pública hashed)
- Após receber a mensagem e utilizando rsaSign() para assinar o valor aleatório recebido, envia uma outra mensagem de volta ao cliente em que cifrando com a chave de sessão lhe envia uma mensagem com o tipo de mensagem('AUTH1'), o valor aleatório assinado e um outro valor aleatório diferente para o cliente assinar.
- Depois do cliente receber a mensagem do servidor, utiliza rsaVerify() para verificar se o valor enviado foi realmente assinado pelo Servidor e envia uma outra mensagem cifrada com a chave de sessão com o tipo de mensagem ('AUTH2) e o valor aleatório novo recebido pelo servidor assinado com rsaSign() e a sua chave pública, para o servidor utilizar no rsaVerify().

Estas duas últimas mensagens enviadas entre servidor e cliente são para garantir a integridade dos dados e assim estabelecer uma sessão entre os dois, assim o servidor

consegue autenticar o cliente/jogador utilizando a assinatura enviada pelo mesmo verificando as chaves públicas.

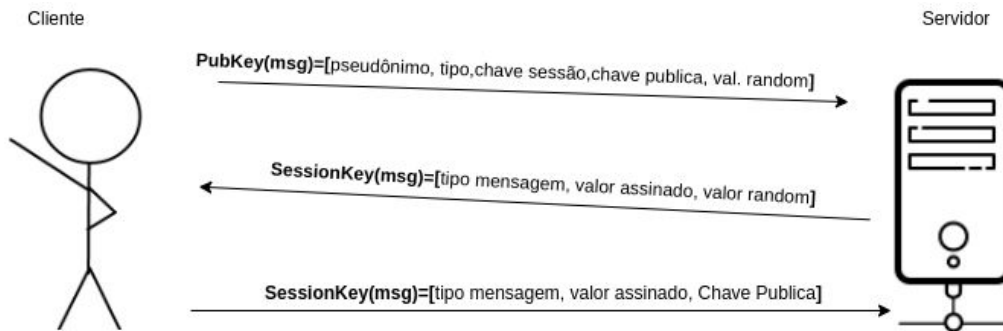


Fig.3: Protocolo de Autenticação Sem Cartão de Cidadão

2.2.2. Protocolo de Autenticação Com Cartão de Cidadão

Para a autenticação com Cartão de Cidadão, a troca de mensagens entre cliente e servidor é idêntica a da autenticação sem Cartão de Cidadão, o que muda apenas é onde o cliente vai buscar a chave pública. Esta chave pública é retirada da leitura do Cartão de Cidadão e de todos os certificados necessários que é feito no `C_Card.py` pela função `getPublicKey()`, sendo que o `C_Card.py` utiliza a biblioteca `"libptcidpkcs11.so"` e o `PyKCS11` e juntamente com certificados necessários consegue-se correr o Cartão de Cidadão e (tendo o pin de autenticação) é obtida a informação do cartão necessária para o projeto, neste caso apenas utilizamos o nome e a chave pública, que obtemos ao extrair o certificado do cidadão após a inserção do pin.

O certificado do cidadão é utilizado para verificar a assinatura do cidadão que para conseguir aceder a assinatura pelo cartão de cidadão é chamando a função `sign()` do `C_Card.py` e irá ser pedido novamente o pin para comprovar que é de facto o dono do Cartão de Cidadão.

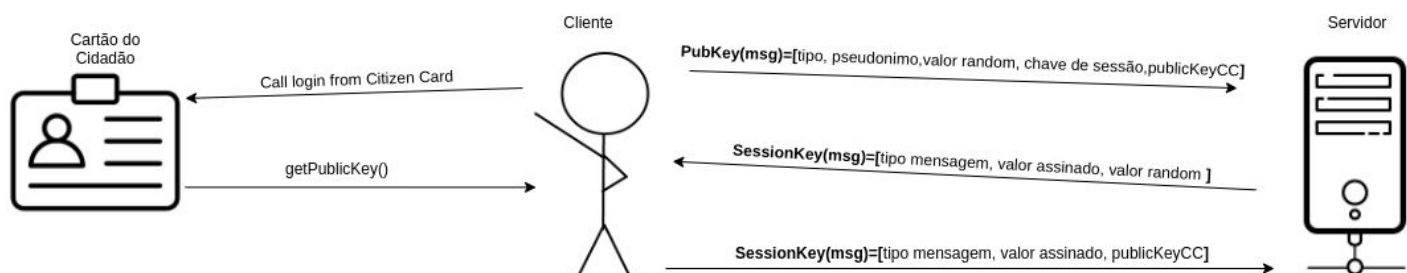


Fig.4: Protocolo de Autenticação Com Cartão de Cidadão

2.3. Pseudonimização da pilha de peças.

Como a pilha de peças tem de passar por cada cliente para ser baralhado, é fundamental que estes não tenham acesso aos valores de cada peça, pois podem manipular a ordem delas de uma forma que a deixe numa ordem mais conveniente para ganhar o jogo. Para se conseguir a integridade do processo de baralhamento o Servidor vai distribuir uma pilha de peças "pseudonimizada".

A "pseudonimização da pilha consiste no Servidor gerar uma chave simétrica (AES(128)) para cifrar cada peça, no fim o Servidor tem à sua disposição uma lista de tuplos da peça pseudonimizada e do seu respectivo índice.

2.4. Protocolo de Baralhar (Randomization Stage)

Após o Servidor ter a pilha de peças pseudonimizada vai distribuí-las ao clientes, para cada um destes a baralhar.

Faz-se uso, novamente, das chaves de sessão de cada cliente para cifrar as mensagens, de envio da lista de tuplos pseudonimizada.

Cada cliente ao receber a lista de tuplos, acrescenta uma camada de cifragem a cada tuplo da lista recebida, para isto, cada cliente gera uma chave simétrica para cifrar cada tuplo.

Agora em detalhe, o Servidor cifra com a chave de sessão do primeiro cliente da sua lista de clientes uma mensagem com um tipo associado ('SHUF0'), e a lista de tuplos e envia para o mesmo. Este decifra com a chave de sessão, gera uma chave simétrica para cada tuplo da lista e cifra-o, ao mesmo tempo que guarda a respectiva chave num dicionário para uso posterior. Com isto foi criada outra lista com cifras de cada tuplo recebido, cada um com a sua chave única. O cliente gera uma mensagem do tipo ('SHUF1'), adiciona a lista recentemente criada de cifras e cifra a mensagem com a chave de sessão que tem com o Servidor, e envia.

O servidor ao receber, repete o procedimento para o próximo cliente, em vez de enviar as pilha pseudonimizado por ele, envia a lista de cifras recebidas pelo cliente anterior, criando assim uma sucessão de cifras em cima de cifras para cada tuplo, da peça pseudonimizada e de seu índice.

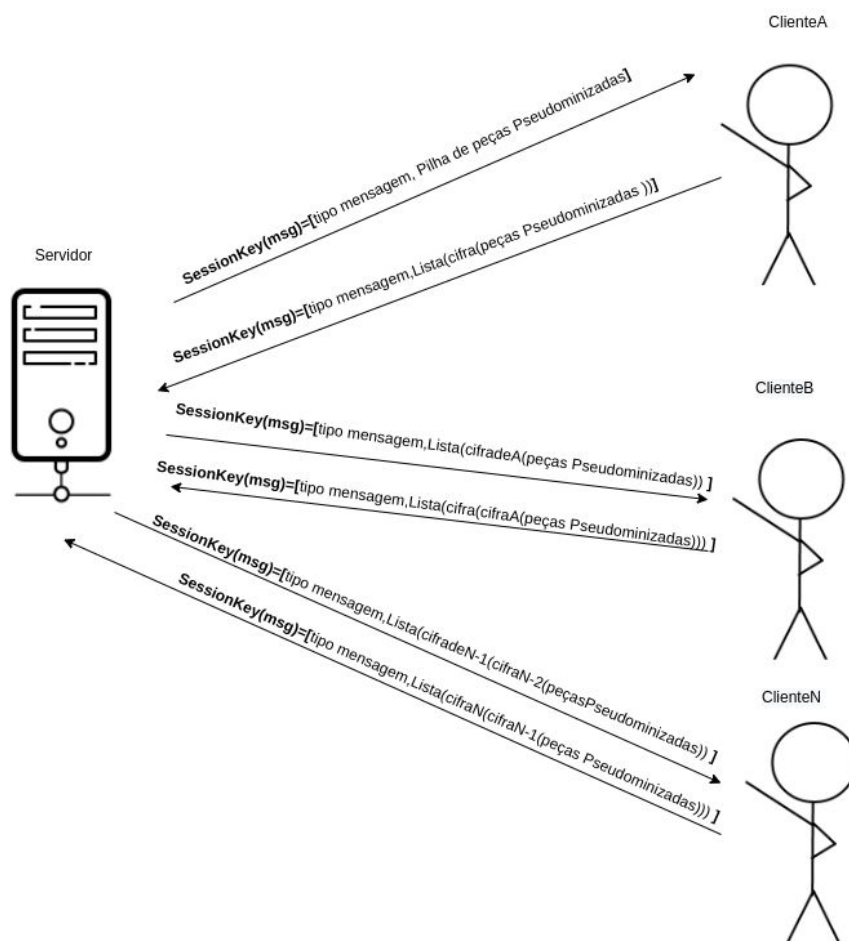


Fig.5: Protocolo de Baralhar

2.5. Protocolo de Seleção (Selection Stage)

O servidor irá agora circular o stock entre os clientes. Para isso irá escolher de forma aleatória um jogador e enviar-lhe as peças encriptadas pelo protocolo anterior através de uma mensagem cifrada com a chave de sessão do cliente. (SEL0)

O cliente ao receber as peças irá decifrar a mensagem com a sua chave de sessão e escolher entre retirar uma peça de forma aleatória do stock que recebeu ou de passar e não fazer nada (probabilidade de passar de 95%).

Se escolher receber uma peça irá retirá-la do baralho, e devolver o baralho já sem a peça encriptada para que seja circulada para outro cliente.

O processo irá acontecer ciclicamente até que cada cliente tiver 5 peças encriptadas na sua posse.

2.6. Protocolo de Comprometimento(Bit Commitment)

Nesta etapa, cada jogador envia um depois do outro, um comprometimento(bit commitment) das peças que tem, ainda cifrado. Todos os jogadores recebem esses bit commitment dos outros jogadores, bem como a pilha inicial de peças no jogo.

Para tal, cada jogador gera 2 valores aleatórios, que são usados para fazer hashing com SHA256 junto com a sua mão inicial que irá comprometer. Depois, este cliente cifra uma mensagem com a chave de sessão com o servidor, mensagem que consiste no seu tipo ("COMM1") e no bitCommitment gerado.

O servidor coleciona todos os bitCommitments (um de cada cliente), junta-os numa lista em conjunto com a pilha de peças inicial do jogo e envia-a a cada cliente(cifrando uma mensagem com a respectiva chave de sessão de cada cliente), o tipo de mensagem ('COMM2') e o primeiro valor aleatório. O segundo valor só é enviado no fim para verificação.

Este compromisso permite que se algum jogador fizer batota, como por exemplo jogar uma peça repetida ou que não se encontra no baralho, seja descoberto no fim de cada ronda.

$$\text{BitCommitment} = \text{SHA-256}(\text{val1}, \text{val2}, \text{peças})$$

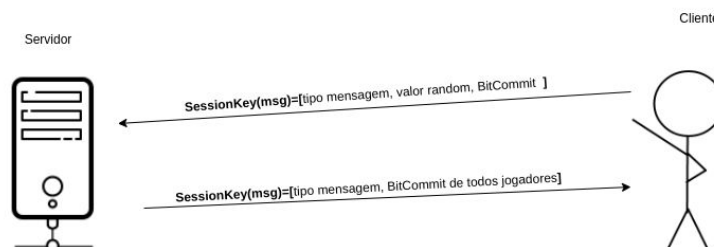


Fig.6: Protocolo de Comprometimento

2.7. Revelation Stage

Nesta fase os clientes irão agora descriptar de forma inversa à usada na Randomization Stage.

Para cada cliente, agora na ordem inversa, irá ser enviada uma mensagem (REVL0), à qual o clientes irão responder com um dicionário, que contém as chave de descriptação guardadas na randomization stage para as peças que já não estão no stock do server.

Após receber esse dicionário, o servidor vai agora enviar esse dicionário para todos os clientes (REVL1), permitindo assim que retirem a primeira camada de encriptação presente nas peças que eles escolheram.

Irá repetir o processo, para cada cliente, de forma inversa, tal como anteriormente referido até cada um ter em sua posse a sua mão, ainda pseudominizada, mas já não encriptada.

2.8. Tile de-anonymization Preparation Stage

De forma a preparar a eventual des-pseudinimização da sua mão, de forma aleatória os clientes irão receber do servidor uma mensagem encriptada com a sua chave de sessão.

Com uma chance de 5%, o cliente irá criar um par de chaves pública-privada.

Após isso, o cliente vai escolher de forma aleatória uma das peças pseudonimizadas em sua posse, tuplo (i, P_i) , e vai preencher um array a ser enviado de volta para o servidor na posição i com a chave pública criada, irá também guardar a chave privada junto com o tuplo (i, P_i) que escolheu.

Irá repetir o processo de forma contínua até o servidor ter em sua posse um array com todas as chaves públicas correspondentes às peças pseudonimizadas nas mãos dos clientes.

2.9 Tile de-anonymization Stage

Por fim nesta fase, o Servidor irá criar uma lista constituída por tuplos de todas as peças des-pseudominizadas conjuntamente com a chave que serviu para as pseudominizar para o cliente.

Esta mensagem irá ser cifrada com a chave pública correspondente da lista formada no passo anterior.

Já no cliente, após descriptação com a chave privada que tinha anteriormente sido guardada no tuplo, o cliente irá usar a chave recebida para de- pseudonimizar cada uma das suas peças e verificar se corresponde à peça recebida pelo server, verificando assim a honestidade de todo o processo.

2.9. Stock Use

Sempre que um cliente não tenha uma peça na sua mão que seja jogável, poderá requisitar ao servidor uma nova peça.

O servidor irá retirar do seu stock de peças de forma aleatória uma peça ainda encriptada e pseudominizada.

Após a descriptação e des-pseudonimização usando os processos referidos anteriormente, irá então encriptar uma mensagem com a chave de sessão do cliente e enviar-lhe assim uma nova peça.

2.10. Contabilidade correta

No fim do jogo, se houver algum jogador autenticado, com cartão de cidadão, é guardado num ficheiro do diretório do projeto, uma mensagem(`saveResult`) que contém o nome do dono do cartão de cidadão assinado com a chave privada para demonstrar que é o

dono do cartão concorda com os pontos, esta informação é retirada do certificado de cidadão, e a pontuação desse jogador. É também verificado se este jogador já jogou algum jogo anteriormente e é assim atualizado o seu score, sendo adicionado ao resultado existente do scores.txt ao do jogo que acabou com os scores anteriores.

```
sign = existCard.infoCC
msg = str(sign)
saveResult = base64.b64encode(existCard.sign(0, bytes(msg, encoding='utf-8'))).decode('utf-8')
self.saveScore(saveResult,points)
```

Fig.7: Assinatura dos pontos

```
score.txt
1 .jux04Amw1c0jnyxTdfq8jAZ0hANV4lg23R0eZEwTCa9hItvi8DaWLW2Jz6YItfw64PtdzoWJVf0UxfNHxUxvV98SCsHQKzM1n0U9sDncrlHsHwoRDkeCbgQUBQ1E5bzRk20KUV3nZtXcEBh0MQGPkxrfYC+wqbGwNC84ElCJl=251
2
```

Fig.8: score.txt , (chave assinada = points)

2.11. Batota

Cada jogador tem a possibilidade de fazer batota dentro do próprio cliente desenvolvido. O cliente “cheater” tem acesso a um baralho diferente do baralho original, e em vez de pedir ao servidor que lhe forneça uma peça quando este não tem como jogar, vai buscar essa peça ao baralho não original que o vai permitir fazer uma jogada inválida, fazendo com que tenha vantagem sobre os outros jogadores.

Implementação

3.1. Problemas conhecidos e limitações

Existiram algumas limitações no desenvolvimento do projeto, bem como a impossibilidade de resolução de algumas features. O tempo que muitas vezes o programa demora a correr, devido ao uso de variadas cifragens bem como a limitação dos computadores pessoais e Sistemas Operativos, tornavam a depuração muitas vezes mais lenta. Não se conseguiu implementar duas partes do objetivo final do projeto, sendo elas reportar se alguém está a fazer batota e Tile de-anonymization Stage. Também não há comunicação direta entre os clientes, visto que optamos por usar o Servidor como intermediário, ao invés de criar uma sessão entre clientes com uso das suas chaves privadas.

- reportar a batota
- tile de-anonymization
- não criamos chave de sessão para comunicação entre clientes

Conclusão

Neste projeto teve alguns problemas e dificuldades , sendo que no decorrer do projeto vários destes problemas foram ultrapassados mas, como referimos anteriormente, nem todas as dificuldades foram ultrapassadas.

Tirando o fato dos problemas não resolvidos, a intenção do projeto foi atingida. Criou-se um jogo de Dominó funcional que inclui as implementações de Segurança e Autenticação, que era a intenção da cadeira de Segurança.

Resumidamente, este projeto ajudou aos seus participantes adquirir e pôr em uso os vários métodos e conhecimentos aprendidos durante as aulas.