universidade de aveiro
Departamento de Eletrónica, Telecomunicações e Informática

# Segurança

## Projeto Prático

2020/2021

***P13***

***Alunos:***

Rodrigo Santos, 89180
Alexey Kononov, 89227
Daniel Lopes, 87881
Rui Santos, 89293

# Introdução

No âmbito da disciplina de segurança do 4º ano do curso MIECT, da Universidade de Aveiro, pretendeu-se desenvolver um sistema que permita aos utilizadores participar num jogo de dominó garantindo o máximo de segurança possível. O sistema de jogo permite o anonimato entre os jogadores e no fim o jogador que ganha recebe os seus pontos provando a sua identidade. O sistema também garante toda a integridade e a confidencialidade das peças distribuídas durante o jogo. Os jogadores usam somente as peças que recebem da mesa, sem a possibilidade de usarem outras peças de maneira ilegal.
Neste relatório encontra-se a descrição e explicação da implementação usada no projeto. Todas as funcionalidades implementadas e medidas de segurança estão detalhadamente descritas nas seções abaixo.

# O Jogo

Este projeto tem como base o jogo Dominó clássico com algumas diferenças. Cada jogador começa com 5 peças numeradas de 0 a 6 e o primeiro a jogar deve ser o primeiro jogador a entrar no jogo. Para uma peça ser corretamente inserida na chain, deve ter um número igual a uma das pontas da mesma. Quando o jogador não tem uma peça para jogar, deve buscar peças da pilha até poder jogar. Quando a pilha estiver vazia e um jogador não tiver uma peça jogável, este deve passar a sua vez. Quando todos os jogadores passam a vez numa ronda, o jogo deve acabar em empate.
Para implementar o jogo, foi retirado algum código do site https://repl.it/@sama/Domino-Game#main.py e colocado no ficheiro python domino.py. Este ficheiro começou por ser a base do início do projeto mas quando se implementaram as sessões TCP, este passou apenas a ser utilizado em poucas partes do servidor. É também utilizado o ficheiro networks.py para auxiliar o ficheiro clientMS.py na gestão das sessões TCP.
A comunicação entre jogadores e Table Manager baseia-se num dicionário que contém diversas informações como um código (dic[‘code’]) que é trocado para indicar as várias intenções dos sujeitos. Estes códigos encontram-se no fim do ficheiro “serverMS.py” com as respetivas descrições.

# Sessions Player-Player, Player-Table Manager

## ● *Player-Table Manager:*

No início a mesa fica à espera dos jogadores até ao número de clientes definido no início, na variável ***nClients***, ou seja essa variável determina o número de jogadores no jogo.

O jogador gera um par de chaves ***RSA 1024 bits***, conecta-se à mesa e de seguida fica num loop à espera da mensagem da mesa com o código da operação. O servidor também gera um par de chaves ***RSA*** e para cada jogador conectado chama a função `get_session_server` no servidor onde o jogador autentica-se e é estabelecida uma session key com o servidor.

Este processo começa com o envio do código "***s00***" para o jogador. O servidor gera 8 bytes aleatórios e calcula a hash deles. Cifra com a chave pública do jogador e o jogador após verificar a integridades dos dados recebidos, calculando hash e comparando com hash recebida, envia de volta 8bytes cifrados com chave pública do servidor. Depois ambos calculam a session key e guardam-na, esta é uma chave simétrica de 128 bits (AES, modo CBC) calculada a partir de 8 bytes gerados no início. O servidor envia um sinal cifrado com a session key estabelecida e caso o jogador consiga decifrar, envia o seu pseudónimo.

***Com CC:***

Caso o jogador tenha o CC, assina o seu pseudónimo com a chave privada do cartão e envia a assinatura junto com o seu certificado. Se o servidor verificar a assinatura com a chave pública extraída do certificado, é comprovada a posse do pseudónimo pelo jogador e a mesa guarda o certificado do jogador que será usado na etapa final de reclamar os pontos. Achou-se esta, a melhor forma de propriedade do pseudónimo vincular o pseudónimo à identidade. Outra alternativa considerada foi cifrar com a chave pública do CC um valor aleatório gerado pelo servidor e no final pedir ao jogador para decifrar com a chave privada do CC e enviar o valor, caso fosse igual confirmava-se ser a mesma identidade.

***Sem CC:***

Sem o CC considera-se o jogador anónimo e no final este não poderá reclamar os pontos ganhos.

## ● *Player-Player:*

Após todos os jogadores estarem conectados à mesa, são estabelecidas *session keys* entre todos os pares de jogadores, que serão usadas para calcular HMAC das mensagens diretas entre jogadores e a confidencialidade quando necessário. Este processo é feito na função `get_session_player(p1,p2)` do servidor. O jogador 1 recebe o código "***s01***" e o

jogador 2 - "***s11***" , seguidamente o servidor apenas encaminha as mensagens entre os jogadores.

O jogador 1 recebe a chave pública de outro, gera 8 bytes aleatórios, assina a hash de 8 bytes com a chave privada e envia para o jogador 2 a mensagem constituída por <8 bytes cifrados com a chave pública do jogador 2 :: assinatura de hash de 8 bytes :: sua chave pública>.

O jogador 2 decifra 8 bytes, calcula hash e verifica a assinatura. Deste modo é garantida a autenticidade e integridades dos dados (assinatura de hash) e confidencialidade perante a mesa (cifra com chave pública do outro jogador). Caso esteja verificada a assinatura, ambos criam uma chave simétrica de 128 bits com algoritmo *AES* no modo *CBC* a partir dos 8 bytes gerados no início pelo jogador 1. Trocam sinal de "ok" cifrado com a chave estabelecida, caso sejam capazes de decifrar dos dois lados, trocam os seus pseudónimos em claro, para internamente associar a chave calculada ao jogador respetivo.

Ponderou-se criar uma função que estabelecia uma chave simétrica temporária entre jogadores apenas quando fosse necessária confidencialidade. Mas pela dificuldade em integrar um mecanismo assim no projeto que se encontrava foi decidido estabelecer chaves de sessão entre jogadores logo no início do jogo.

Para implementação das chaves simétricas existem imensas alternativas e foi usado algoritmo *AES* com o modo de cifra *CBC* por considerar mais simples. A troca das chaves é feita com *RSA Encryption.*

# Protection of the messages exchanged:

O controlo de integridade e autenticidade das mensagens é garantida com Hash-based Message Authentication Codes. Optou-se por este método porque existem session keys estabelecidas entre, mesa e jogadores e entre jogadores. Sendo usada a chave simétrica o processo é rápido e bastante seguro. Tanto no servidor como nos clientes, as funções de enviar e receber as mensagens recebem como argumentos a mensagem a ser enviada e a *session key* estabelecida com a entidade com qual é feita a comunicação.

Na função de envio é calculado resumo do HMAC usando a session key o algoritmo SHA1 e é concatenado à mensagem. A mensagem final enviada tem o seguinte formato: <message :: hmac(message) > .

Na função de receção, a mensagem é dividida em mensagem original e resumo HMAC.

É calculado o resumo HMAC a partir da mensagem original e é comparado ao recebido. Se for igual, a autenticação e controlo de integridades estão verificados. Foi usado o operador == na verificação de digest em vez de `compare_digest()`, o que pode aumentar a vulnerabilidade aos timing attacks.

Quando é requerida a confidencialidade das mensagens, o conteúdo é cifrado com a chave de sessão.

# Deck secure distribution protocol

Nesta fase do jogo, após tomadas as precauções e medidas de segurança necessárias, será feita a distribuição das peças iniciais de cada jogador. Este processo é longo, e por isso está dividido em 7 partes.

- ***Pseudonimização das peças:***

Para desenvolver o protocolo de distribuição segura das peças, começou-se por aplicar o primeiro ponto descrito no guião do projeto, onde, para a primeira fase da pseudonimização das peças, por parte do Table Manager (server), é criado em primeiro lugar um array de peças contendo todas as peças, e de seguida esse mesmo array é baralhado.

Para cada uma das peças baralhadas, é adicionado um índice, do qual resulta uma peça, no formato [**x**, (**y**, **z**)] em que **x** corresponde ao valor do índice e **y, z** correspondem aos valores da peça.

Posteriormente, no mesmo ciclo, é gerada uma chave aleatória com 16 caracteres, sendo que esta é armazenada em conjunto com o índice da peça a que pertence no array *indexKeys*.

Com a chave é gerado o pseudónimo da peça que é criada, fazendo hash, do resultante da junção do índice da peça com a chave e os valores das peças, utilizando a função `hash()` de python. Este pseudónimo é guardado em conjunto com o índice num array "*PseudonymizedDominos*", para referência futura na fase de "*Tile De-anonymization*".

- ***Randomization Stage:***

Na fase seguinte, "*Randomization Stage*", o deck das peças pseudonimizadas é enviado para o primeiro jogador que se ligou ao Table Manager, este jogador e todos a seguir, fazem a cifra do deck baralhando-o após a sua cifra.

A cifra é feita com recurso a uma cifra simétrica AES no modo CFB. Para cada peça é gerada uma chave com 16 caracteres aleatória e um IV também aleatório, tanto o IV como a chave são armazenadas em triplos em conjunto com a cifra resultante da cifra num array denominado de "*keysUsed*", para uso na "*Revelation Stage"*.

Para garantir o correto funcionamento, é feita a verificação da existência da cifra resultante da cifra no set de peças, caso exista já uma cifra igual, esta é recalculada.

- ***Selection Stage:***

Após a cifra das peças, o deck das peças cifradas, *dic['dominoSet']*, assim como um array de strings ('True', 'False') com o mesmo tamanho do array de peças, *dic['chosenDominos']*, circula pelos jogadores, e cada jogador quando recebe o deck, gera um número de [0 a 100], caso o número gerado esteja entre 0 e 5 (5%) o jogador gera outro número de [0 a Número Total de peças] para fazer a seleção de uma peça, caso esta não esteja já selecionada, esta seleção é marcada no array *dic['chosenDominos']* no qual cada valor, corresponde a uma peça, em que:

*'False' → not taken.*
*'True' → taken.*

Depois de tomada a decisão sobre escolher ou não uma peça, o jogador envia para outro jogador, random, o deck e o array de escolhas, sendo que este último vai cifrado com a chave simétrica entre esses dois jogadores para garantir confidencialidade e também é calculado HMAC do array de escolhas, que é concatenado com o próprio array de escolhas para garantir integridade.

- ***Commitment Stage:***

Assim que um jogador preenche a sua mão, ou seja, seleciona 5 peças, gera o Bit Commitment da sua mão inicial e envia o Bit commitment e um dos valores random usados (R1) para o Table Manager. Cada jogador gera dois valores R1 e R2, aleatórios, pertencentes ao intervalo [1000, 5000], e é efetuado um hash do resultado da concatenação dos dois valores aleatórios e a mão inicial de cada jogador, utilizando a função `hash()` de python. Esta fase acaba quando o Table Manager recebe um conjunto [Bit commitment, R1] de cada jogador com o qual tem ligação.

De seguida o Table Manager envia os conjuntos [Bit commitment, R1] de todos os jogadores para todos os jogadores, e estes armazenam esses commitments, concluindo assim a fase de Commitment Stage.

- ***Revelation Stage:***

Na fase de Revelation, o Table Manager envia um array com o tamanho igual ao número total de peças para a escrita das chaves e IV necessários à decifra, o array de valores boolean da seleção de peças e o deck cifrado atual, para o último jogador que cifrou o deck, este jogador verifica que peças foram escolhidas no array de peças selecionadas, *dic['chosenDominos']*, e para cada uma verifica e escreve no array de chaves, *dic['keys']*, a chave e o IV que usou para fazer a cifra de uma determinada peça consoante a sua cifra. O jogador reenvia os elementos que recebeu, com o array de chaves preenchido nos índices das peças escolhidas pelos jogadores, para o Table Manager e este faz circular de seguida o array de chaves e o deck, para decifra das peças escolhidas por cada jogador. Cada jogador apenas faz a decifra das peças que escolheu e faz a substituição do resultado da decifra no deck de peças que recebeu. Assim, no final da decifra o deck contém as peças que todos os jogadores escolheram decifradas em um nível. Este processo é repetido, para todos os outros jogadores, pela ordem contrária à qual foi efetuada a cifragem. No final deste processo todo, o deck contém todas as peças que foram escolhidas na forma pseudonimizada e as outras permanecem cifradas, e cada jogador contém apenas as peças que escolheu na fase de seleção na forma pseudonimizada também.

- ***Tile de-anonymization preparation Stage:***

Para a fase Tile de-anonymization preparation stage, os jogadores devem passar entre eles uma lista onde escrevem nas posições das peças que escolheram, uma chave pública para a qual têm uma chave privada, para cada peça.

Neste processo, o jogador tem 5% de probabilidade de inserir uma chave e 95% de passar a lista para outro jogador. É também nesta fase que se faz uso das sessões entre jogadores criadas no início do jogo. Através destas sessões, o jogador deve escolher aleatoriamente o próximo jogador a enviar a lista, usar a chave simétrica gerada para aquele jogador e cifrar a lista de chaves públicas. É imperativo que esta lista não fique visível para os outros jogadores e para a mesa, caso contrário, os agentes iriam saber que a peça de índice x pertence ao jogador que enviou a mensagem. Assim, é enviado o dicionário que contém a lista para o Table Manager que deve encaminhar a mesma para o jogador destino, cujo pseudónimo se encontra no *dic['player']*. É necessário que o Table Manager troque o pseudónimo no *dic['player']* para o pseudónimo do jogador que está a enviar a mensagem. Caso contrário, o jogador destinatário não conseguirá decifrar a lista de chaves públicas a menos que tente com todas as chaves de sessão.

Quando um jogador verifica que existem cinco vezes o número de clientes, é porque todos os jogadores inseriram as chaves nas respectivas posições. Nesta fase, o jogador coloca o valor de *dic['nPlayersReady']* a True para que o server possa passar para a Tile de-anonymization stage.

- ***Tile de-anonymization Stage:***

Após todos os jogadores terem preenchido a lista com as suas chaves públicas, o Table Manager dá início à última fase do protocolo - Tile de-anonymization stage. Começa por fazer a de-anonimização das peças selecionadas, verificando a que peça corresponde determinado índice, e cifra cada uma com a chave pública correspondente do array de chaves públicas preenchido anteriormente. De seguida envia as peças cifradas para todos os jogadores e cada um decifra as suas peças utilizando as chaves privadas correspondentes. Assim, no final desta fase todos os jogadores têm acesso às suas peças de forma legível e está tudo preparado para dar início ao jogo.

# Stock Access

Após serem escolhidas as peças iniciais (mão inicial) de cada jogador, as restantes peças são armazenadas, ainda cifradas, num array que será a Stock do jogo. Quando um jogador, se encontra no seu turno, e não tem peças na mão com as quais possa jogar, este jogador pode fazer um pedido de acesso à Stock ao Table Manager. Este envia ao jogador a Stock, caso ainda tenha peças, caso contrário o jogador será obrigado a passar o turno.

Ao receber a Stock o jogador retira uma peça random da Stock, fazendo o seu *pop.* Uma vez que esta peça se encontra cifrada, é necessário refazer o processo de decifra correspondente aos passos ***Revelation Stage*****,** ***Tile de-anonymization preparation Stage*****,** bem como a fase ***Tile de-anonymization Stage*** já mencionados na secção ***Deck secure distribution protocol*** deste relatório. O processo é no entanto um pouco simplificado, uma vez que apenas é feito o processo para uma peça.

O processo inicia com a fase de revelação, na qual, uma vez mais, os jogadores revelam a chave que usaram para cifrar a peça específica escolhida da Stock. Os jogadores são percorridos pela ordem inversa de cifragem e quando revelam a chave que usaram para cifrar a peça, esta é decifrada, e enviada para o jogador que a cifrou antes do jogador atual. Até que o jogador que acedeu à Stock tenha a peça na forma pseudonimizada.

Concluída esta fase, o jogador gera um par de chaves, pública e privada, para a peça, e envia essa chave pública para a mesa.

O Table Manager efetua a de-anonimização da peça e cifra a com a chave pública recebida. De seguida envia a peça cifrada com a chave pública para o jogador correspondente. O jogador faz a decifra da peça e adiciona-a à sua mão.

Caso a peça que tenha retirado possa ser jogada, o jogador pode utilizá-la, caso contrário fará novo acesso à Stock, repetindo se o processo.

# Possibility of cheating

Foi introduzida a possibilidade dos jogadores poderem fazer batota. Isto acontece quando, na sua vez de jogar, o jogador pode introduzir a letra C e inventar uma nova peça. Seguidamente, a peça é enviada ao Table Manager e é iniciado o processo de validação da jogada.

# Validation of Tiles & Protest against cheating

Depois de um jogador enviar a peça que pretende jogar ao Table Manager, este deve verificar se a peça se pode inserir num dos extremos da chain. Caso contrário, a peça é rejeitada e é pedido ao jogador que jogue outra vez. No caso da peça já existir na chain, o Table Manager deve rejeitar a peça. Quando a peça jogada tem números de pontos inferior a 0 ou superior a 6, o Table Manager considera a peça inválida.

Depois da validação da peça por parte do Table Manager, este deve perguntar a todos os jogadores se concordam com a peça jogada. Os jogadores verificam se a peça jogada já existe na mesa ou na sua mão e, no caso de um destes se verificar, devem acusar que existe batota ao Table Manager. Neste caso, o jogo deve acabar com um empate, ou seja, ninguém recebe pontos.

# Game Accounting and Claim of points

Após a conclusão da partida inicia-se o processo de atribuição dos pontos aos clientes, para o cliente que ganhou a partida é lhe atribuído dez pontos, para empate é atribuído zero pontos a todos os clientes e zero pontos também para os que perdem a partida.

Se o pseudónimo não provou ser propriedade de uma identidade durante o registo, não é possível reclamar os pontos ganhos.

Portanto antes do servidor atribuir os pontos ao cliente que ganhou a partida tem que se desencadear uma série de etapas.

A primeira etapa consiste em enviar uma mensagem sobre o formato JSON para o cliente que ganhou, contando as informações do jogo. Estas informações consistem na data do jogo, jogadores que participaram e os seus respetivos pontos.

Antes de enviar essa informação, é gerada e guardada um hash dessa mesma mensagem. A mensagem é então enviada para o cliente e este ao a receber, verifica se realmente tem os seus pontos atribuídos. Para isso e baseado no seu pseudónimo, verifica os pontos que

estejam associados. Após essa verificação geram-se duas assinaturas com a chave privada do CC, primeira do JSON com a informação do jogo e a outra do pseudónimo.

Após as assinaturas estarem concluídas são enviadas para o servidor juntamente com a mensagem sobre a informação do jogo. O servidor quando recebe a informação do jogo verifica se o hash corresponde ao anteriormente feito, isto garante que não houve nenhuma alteração dos dados originais do jogo pelo jogador.

O servidor verifica a assinatura do pseudônimo recebida nesta etapa com o certificado do jogador que guardou no processo de registo antes do jogo. Foi feito assim a pensar que desta maneira é garantido que a identidade que pretende reclamar os pontos é a mesma que registou o pseudônimo no início do jogo.

Sendo que tudo esteja normal, é criada uma pasta "*accounting*" no diretório do servidor caso não exista, para posteriormente escrever lá um ficheiro com os dados do jogo e a assinatura do jogador que ganhou a partida. Desta maneira é possível uma identidade consultar o seu histórico de jogos percorrendo os ficheiros e verificando as assinaturas, aquelas que conseguir verificar são os jogos que essa identidade ganhou. Considerou-se que assim a assinatura serve como "id" da identidade sem a revelar.

# Conclusion

Com este projeto podemos pôr em prática vários métodos e matérias distintas lecionadas nas aulas teóricas e práticas de proteção, autenticação e cifragem/decifragem de mensagens, bem como de assinatura de mensagens com recurso a smart cards. Vários algoritmos foram utilizados, e como tal alguns problemas surgiram do seu uso, fomos desafiados a ultrapassá-los e com isso aprofundamos o nosso conhecimento acerca destas matérias.

Todas as funcionalidades de segurança propostas foram implementadas, no entanto, é do nosso conhecimento que a verificação de batota não está a ser feita de forma 100% correta uma vez que não é feita a verificação do Bit Commitment nessa fase

Não foi implementada nenhuma validação da cadeia de confiança (chain of trust)

# References

Algumas ideias para as comunicações seguras:

https://medium.com/@md.julfikar.mahmud/secure-socket-programming-in-python-d37b93233c69

Código domino.py:

https://repl.it/@sama/Domino-Game#main.py