Relatório

Para o desenvolvimento do projeto decidimos desenhar a nossa solução recorrendo ao uso de *Stubborn Links* e *Best-Effort Broadcast*.

Escolhemos usar *Stubborn Links* em primeiro lugar por garantirem que se um dado processo **pi** envia uma mensagem **m** a um processo correto **pj**, então **pj** entrega **m** um número infinito de vezes.

Depois pelo facto do uso de *Stubborn Links* também nos garantir que não são entregues mensagens que não foram criadas.

Na nossa abordagem de implementação dos Stubborn Links o processo principal lança threads secundárias que estão encarregues de enviar as mensagens aos outros processos e esperar a receção de um acknowledge. Para evitar sobrecarregar o sistema, recorremos ao uso de um timeout que é o tempo que esperamos para reenviar a mensagem enquanto não for recebido o acknowledge.

Vale a pena mencionar que por cada vez que reenviamos a mensagem aumentamos o tempo deste timeout.

Optámos por não usar *Perfect Links* porque não nos interessa garantir a propriedade PL1: *Validity*, dado que o algoritmo que seguimos é sequencial e não nos interessa que os processos entreguem todas as mensagens que recebem (p.ex: caso dos PREPERARES).

Também não utilizámos os *Authenticated Perfect Links* dado que não nos interessa garantir autenticidade. Esse fator já é garantido de outra forma como iremos explicar mais detalhadamente adiante.

A opção do uso do *Best-Effort Broadcast* partiu desde logo por este tipo de protocolo nos dar várias garantias como: validade, não duplicação e não criação.

Para além disso, analisando o algoritmo *The Istanbul BFT Consensus Algorithm* garante-nos propriedades que nos permitem escolher este algoritmo ao invés de outros como o *Reliable Broadcast*.

O *Reliable Broadcast* garante que para qualquer mensagem m, se um processo correto entrega m então todos os processos corretos entregam m. No nosso caso não poderemos fazer com que isso aconteça, dado que no algoritmo inicialmente o processos líder envia um pedido PREPREPARE aos que não são líderes recebam do líder e em seguida estes devem enviar pedidos PREPARES e não PREPREPARES. Ou seja, não devem fazer *broadcast* da mensagem enviada pelo líder mesmo esta tendo sido entregue por um processo correto.

O mesmo se aplica em relação à propriedade BCB4 do algoritmo *Byzantine Consistent* Broadcast. É algo que não nos interessa garantir, uma vez que apenas pretendemos assegurar que cada processo correto entrega as suas próprias mensagens.

Achámos que a escolha dos *Stubborn Links* era melhor em relação a termos escolhido *Authenticated Perfect Links* visto que as mensagens trocadas já se encontram autenticadas através da cifra de um token enviado na mensagem com recurso a uma chave privada do emissor. Iremos explicar mais detalhadamente este processo mais à frente.

Em relação aos mecanismos de proteção, todas as mensagens trocadas entre os servidores e entre os servidores e o cliente estão contidas num *Json Object* que se encontra encriptado com uma chave secreta (seguindo o algoritmo AES com o modo ECB) partilhada entre os dois lados da comunicação.

Esta chave foi gerada a partir dos comandos fornecidos no primeiro *lab* da cadeira.

De forma a garantir a autenticidade das mensagens, dentro de cada *Json Object* encontra-se um *token* que está cifrado com a chave privada do lado do emissor e posteriormente poderá ser decifrado do lado do recetor recorrendo ao uso da chave publica do emissor, previamente disponibilizada.

Neste tipo de cifra pública-privada utilizados o algoritmo RSA.

As chaves usadas para o bom funcionamento do protocolo encontram-se presentes na diretoria *SEC-Project/Secure-Messages/keys*. Vale a pena referir que idealmente cada comunicação diferente possuiria uma chave secreta exclusiva para essa comunicação que não seria conhecida pelas comunicações alheias.

Cada processo (cliente/servidor) possui um par de chaves pública-privada sendo que a chave privada é de conhecimento exclusivo ao seu detentor e a chave pública conhecida por todos aqueles que vão comunicar com esse processo.

De modo a verificar a integridade fazemos um digest da mensagem cifrada (payload) com o algoritmo SHA3-256 e ciframos o resultado deste digest (mac) com a chave secreta. Quando a informação chegar ao outro lado da conexão, deciframos o mac com a chave secreta que é partilhada pelos 2 lados da conexão e em seguida fazemos o digest da informação (payload) e comparamos com o mac recebido.

Descartamos mensagens iguais provenientes de processos com o mesmo port durante a mesma instância de consenso como forma de prevenir replay attacks.

Para evitar impersonation attacks na comunicação entre cliente e servidor usamos assinaturas digitais nos pedidos enviados entre os mesmos.

Concluindo, com o algoritmo implementado qualquer cliente que se ligue ao serviço da blockchain tem a garantia de que a sua string será adicionada à blockchain e que esta operação não poderá ser repudiada.