Sistemas Distribuidos **Projecto Conjunto ES+SD** **Grupo n.º 23**

Marco Pereira – 70644

Nuno Simões – 70568

Rita Pereira – 70473

SD-STORE.A

Sendo esta parte, a mais complicada e elaborada de implementar decidimos dividir a tarefa em 3.

Na primeira parte decidimos implementar a replicação dos servidores, para isso efetuamos alteração à *pom* e ao *StoreMain* do *SD-Store*. Na *pom* dividimos o URL dos N servidores em 3 argumentos para que na altura do seu lançamento sejam publicados os caminhos dos servidores corretamente.

Na segunda parte criamos uma Tag, composta por um número de sequência associado a cada servidor. Esta Tag será enviada juntamente com o conteúdo dos pedidos e respostas entre os servidores e os clientes, através de mensagens SOAP num ficheiro XML, posteriormente analisadas nos *handlers*.

Na terceira e ultima parte da tarefa, implementámos o protocolo *quorum consensus.* Para isso realizamos alterações na camada front-end do cliente e na classe StoreImpl do servidor.

- Na parte do cliente, nomeadamente na função 'Load', enviamos para os vários servidores um pedido de leitura, pedido esse que irá retornar pelo menos a maioria das leituras juntamente com a Tag, do conjunto dos servidores. Dessas respostas será enviada ao cliente o conteúdo a que estiver associado o maior número de sequência. Na função 'Store', ao inicio será feito um pedido de 'Load' a um utilizador especifico, apenas para receber o número de sequência associado a cada server. Seguidamente será enviado um pedido de 'Store' com o conteúdo pretendido para armazenar na parte SD-Store juntamente com a Tag, composta pelo número de sequência obtido +1, esperando pelo ack na resposta da maioria dos servidores .

- Na parte do servidor, a função 'Load' apenas irá associar o conteúdo do documento armazenado, ao número de sequência do servidor e enviar a resposta ao cliente. A função 'Store' irá verificar primeiro se o número de sequência recebido na Tag é maior do que o atual e em caso afirmativo irá substituir ou adicionar o conteúdo ao documento e substituir a Tag pela recebida, respondendo com ack ao cliente.

Já que não existe uma maneira nativa que funcione nos vários sistemas operativos de forma igual, decidimos implementar o pdi como número random associado à Tag. Assim o protocolo *quorum consensus* fica completo, com o desempate em caso de igualdade no número de sequência.

TÉCNICO LISBOA

SD-ID.B

Na segunda parte do projeto criamos uma camada de segurança no Cliente, uma vez que é no cliente que são invocados os métodos de 'Store' e 'Load'.

Nesta camada de segurança começamos por implementar a geração da Key que cada cliente irá possuir para cifrar e decifrar os seus documentos. Esta Key é gerada na função *SecretKey generateKey()* através do algoritmo AES, este foi escolhido por ser o sucessor do DES e por ser atualmente o algoritmo de cifra mais seguro, e com 128 bits. Este número de bits é o ideal pois é seguro o suficiente para a Key não ser quebrada e não torna a aplicação demasiadamente pesada.

Após a geração da Key, usamos também no nosso algoritmo de cifra e decifra uma IV-Key, gerada na função *IvParameterSpec generateIv()*. Esta IV-Key tem também 128 bits e serve para criar aleatoriedade nos bits cifrados na chamada da função 'Load'.

Depois da geração das Keys necessárias, na função 'Store' ciframos o array de bytes recebido com ambas as Keys geradas, através do algoritmo AES, do modo CBC (cadeia de blocos cifrados) e da API PKCS5PADDING (cria blocos com o tamanho de 8-bytes). Esse array de bytes será posteriormente armazenado no SD-Store. Na função 'Load' primeiro obtemos a informação cifrada no SD-Store, e só depois será decifrado o array de bytes, através do processo inverso, usando também o algoritmo AES, o modo CBC e a API PKCS5PADDING.

Como toda a informação no SD-Store se encontra cifrada de forma segura e consistente, não são precisos módulos de segurança adicionais.

O único requisito de segurança que não foi concluído, foi a implementação do código de autenticação (MAC).