TÉCNICO LISBOA

Sistemas Distribuídos

2º Semestre 2014/2015 (Taguspark)

Grupo 59:

Joana Condeço
nº 68624

Ana Salta
nº 74254

Miguel Pires
nº 76433

1 SD-ID.A

Ao implementar a solução para o primeiro requisito tivemos de tomar algumas decisões relativamente à segurança e representação de dados:

Algoritmo de cifra - para satisfazer a necessidade de manter a comunicação entre clientes e serviço confidencial, decidimos utilizar o algoritmo AES com modo de operação CBC. Esta escolha deveu-se ao facto de que o algoritmo AES se tornou num *standard* na indústria e é aceite como sendo o mais seguro e ao facto de que o modo EBC iria enfraquecer o nível de segurança oferecido. (ELABORAR).

Tamanho de chave - tomada a decisão de usar o algoritmo *AES* tivemos de optar entre utilizar chaves de 128, 192 ou 256 bits. Escolhemos utilizar chaves de 128 bits por motivos pragmáticos. Para gerar chaves de 192 ou 256 bits era necessário descarregar e configurar o ficheiro "*Unlimited Strength Jurisdiction Policy Files*" e decidimos não o fazer dado que se o projecto fosse corrido numa máquina sem esta configuração ocorreria uma excepção. Para simplificar, são geradas apenas chaves de 128 bits mas para usar chaves de 192 ou 256 bastaria configurar o ficheiro anteriormente mencionado.

Geração de chaves baseadas em *passwords* - para gerar chaves AES que fossem baseadas nas *passwords* dos utilizadores foi usada a função de derivação PBKDF2. Apesar de esta escolha acarretar algumas complicações adicionais quando comparada com funções mais simples (SHA-1, SHA-2, etc), achámos que seria uma opção mais segura. Uma das complicações introduzidas é a necessidade de incluir *salt* no formato de um *byte array*, que é baseado no *username* do utilizador. Embora seja possível argumentar que este valor devia ser escolhido aleatóriamente e que para qualquer atacante, que conheça a função de *hashing* usada, o *username* seria uma escolha óbvia para qualquer atacante, também é possível argumentar o contrário. O propósito da utilização de *salt* não é só aumentar o tamanho da chave mas também

evitar um ataque através de *rainbow tables*[1] em que um atacante utiliza tabelas pré-computadas de *hashes passwords* para quebrar a senha. Ao utilizar um salt, mesmo que este seja uma variação do username, estamos a impedir um atacante de utilizar estas tabelas, obrigando-o fazer toda a computação necessária a um *brute-force* e, por fim, tornando o ataque inexequível.

2 SD-STORE.B

Para a implementação da solução do segundo requisito de assegurar tolerância a faltas, decidimos tomar as seguintes decisões:

Desenho do protocolo - Para as operações *createDoc* e *listDocs* reparámos que a solução ideal seria um simplificação do protocolo *quorum consensus*. Nesta versão simplificada não é necessário associar *tags* a cada versão do objecto a guardar, dado que, como não existe uma operação de remoção nunca irá ocorrer o caso em que um documento não é removido numa réplica e uma nova versão do documento é posteriormente adicionada, deixando assim duas versões diferentes no sistema. Se fosse possível esta situação ocorrer, teriam de ser utilizadas *tags* para ser possível distinguir versões distintas do mesmo documento.

Consistência - Feita a análise anterior, decidimos implementar um protocolo em que uma escrita ou leitura deixa o servidor consistente quando são recebidos *Write Threshold* (WT) ou *Read Threshold* (RT) *acknowledgements,* respectivamente. Para assegurar a consistência eventual pretendida, é executada uma fase de *write-back* após cada operação *listDocs* . Esta fase cumpre o propósito de tentar que todas réplicas estejam actualizadas.
Para ser garantida a consistência, as seguintes equações têm de ser sempre verdadeiras:

- 2WT > REPLICAS_NUMBER
- RT + WT > REPLICAS_NUMBER

Sincronismo - Apesar de não ser directamente útil para a funcionalidade deste projecto, decidimos tornar os pedidos em pedidos assíncronos com o recurso a *callbacks*. Esta solução tem a vantagem de potencialmente não implicar *polling* (espera activa) e consequentemente obter melhor *performance*.

[1] http://en.wikipedia.org/wiki/Rainbow_table