INSTITUTO SUPERIOR TÉCNICO – TAGUSPARK

Sistemas Distribuídos - Relatório do 2º Projecto

Grupo 67 – Turno: Quarta, 9:30 – 0.14

Marco Flávio dos Santos Tomás, Nº 65921
LEIC-T

Vanessa Filipa Casado Gaspar, Nº 73995
LEIC-T

David Pereira da Silva, Nº 76511
LEIC-T

Requisito SD-ID.B

A solução implementada oferece garantias de confidencialidade, autenticidade e integridade sobre os conteúdos dos documentos que são obtidos a partir do SD-STORE. A implementação da solução foi dividida em 4 etapas:

1. Geração da chave secreta (algoritmo de cifra simétrica AES);
2. Codificação da chave e conteúdo (método BASE64);
3. Utilização da assinatura digital (algoritmo MAC);
4. Codificação do conteúdo armazenado (algoritmo de cifra simétrica AES).

O algoritmo escolhido para a geração da chave foi o AES, devido à sua simplicidade de utilização e também pelo facto de não exigir um processamento demorado, pois não realiza operações aritméticas. A chave gerada possui 128 bits e não 256 bits, tal como foi ponderado pois aumentava a segurança, dado que seria necessário instalar *Unlimited Strength JCE Policy Files*.

Quando o cliente realiza a operação *Store*, o servidor codifica a chave e o conteúdo, utilizando o método BASE64 e de seguida, inicia o algoritmo MAC, que foi escolhido por conferir autenticidade e integridade à solução. Começa por utilizar o algoritmo de *hash* MD5, com a finalidade de originar o *digest* do conteúdo codificado concatenado com a chave codificada. O *digest* é armazenado

localmente e por fim, o servidor encripta o conteúdo a enviar utilizando o algoritmo AES, modo ECB e *padding* PKCS5Padding, o que confere confidencialidade dos dados no transporte na rede.

A decisão da chave e o conteúdo serem codificados antes da criação do *digest*, consistiu no facto de aumentar a segurança perante um ataque, dado que o *digest* é armazenado localmente.

Deste modo, mesmo que o atacante consiga decifrar o *digest*, não seria possível obter o conteúdo e a chave utilizada, aumentado o nível de confidencialidade dos dados.

Quando o cliente realiza a operação *Load*, o servidor desencripta o conteúdo recebido, codifica a chave e conteúdo, utilizando o método BASE64 e de seguida, realiza o *digest* do conteúdo concatenado com a chave e compara o novo *digest* com o *digest* originado na operação *Store*. Se ambos os *digest* forem diferentes entre si, significa que o conteúdo foi modificado e não é retornado ao utilizador, garantido a autenticidade e integridade dos dados.

Requisito SD-Store.A

A tolerância a faltas é crítico e por isso a abordagem utilizada foi a replicação activa que se caracteriza pela existência de múltiplos servidores (gestores de réplica) que recebem os pedidos dos clientes pela mesma ordem, efectuam a operação, e respondem ao cliente assim que obtiver uma maioria de respostas.

Na implementação da solução foram garantidos os seguintes pontos:

- Consistência sequencial;
- Transparência de replicação;

O cliente cria uma instância do *FrontEnd* e envia-lhe os seus pedidos. Por sua vez, o *FrontEnd* realiza *lookup*, com a finalidade de encontrar todos os gestores de réplica registados (no caso do projecto, são 3 servidores). O cliente nunca interage directamente com os servidores replicados, pelo que existe transparência de replicação.

Cada documento possui um campo *tag*, constituído pelo número de sequência e o identificador do cliente e é transportado através de *handlers criados para o efeito, um para o cliente e outro para o servidor*.

As operações *Store* e *Load* implementam o protocolo *Quorum Consensus*, variante ABD (com writeback), pelo que quando o cliente realiza a operação *Load*, o *FrontEnd* envia o pedido a todos os servidores, para que, caso tenha havido alguma falha em algum deles, estes executem e alterem os dados das suas bases de dados, enviando para o *handler* a *tag* do documento e cliente através do header da mensagem. Após a execução do pedido, todos os servidores ficam consistentes.

De forma a melhorar toda a comunicação e torna-la mais rápida, foram usadas chamadas assíncronas. O servidor, ainda para manter consistência, faz inicialmente um read() por forma a obter o valor da maior tag existente, por forma a comparar com a sua e actualizar o seu sequence number.