Dependable Public Announcement System

SEC
Grupo 18

Carolina Carreira 87641 Catarina Pedreira 87524 Miguel Barros 87691

1. Explicação do Design

O sistema tem uma arquitetura cliente/servidor dividida nos seguintes módulos:

Gestor de Persistência

Garante que não há perda ou corrupção de dados quando ocorre um crash do servidor (secção 3).

Gestor de Sessões

Garante a integridade e frescura dos pedidos que chegam ao servidor(secção 4).

API do servidor

A API recebe pedidos do front-end e responde ao mesmo.

Front-End

O Front-End faz pedidos ao servidor em nome dos clientes e gere as sessões dos mesmos.

Aplicação Cliente

A aplicação recebe pedidos do utilizador e envia-os para o front-end.

Infraestrutura de chaves

Todas os pares de chave são RSA-512.

As chaves públicas e privadas estão numa *keyStore*, existindo uma para a aplicação cliente e uma para o servidor. Todas as chaves privadas estão protegidas por password. A keystore do cliente contém a chave pública do servidor.

Figura 1. Flow de um pedido bem-sucedido

2. Garantias de integridade

- **Deve ser possível provar que um post foi criado por um dado utilizador.**
  Todos os posts contêm uma assinatura digital do seu conteúdo (mensagem, autor, board destino e referências) que só pode ser gerada com a chave privada do autor. O servidor verifica as assinaturas de todos os posts recebidos.

- **Um utilizador só deve conseguir postar no seu board ou no board geral.**
  Cada board particular tem associada a chave pública do seu utilizador. Para um post ser publicado num board, o seu conteúdo deve ser assinado com a chave privada correspondente.

- **Nenhum utilizador deve conseguir fazer um post em nome de outro.**
  Para criar um post, o utilizador deve assiná-lo corretamente, provando conhecer a chave privada do autor, que é única e secreta.

- **Deve ser possível provar que um post tinha como destino o board geral ou o board do utilizador.**
  Para isto, podemos verificar a assinatura digital do post que contém um identificador único do board a que foi destinado.

- **Inputs anómalos devem ser detectados pelo servidor, que deve gerar excepções apropriadas e retorná-las ao cliente.**
  Se existir um erro (ex: uma assinatura inválida) o servidor gera uma excepção e retorna-a ao front-end.

3. Outras garantias de confiabilidade

Persistência e Integridade de Dados

- O servidor guarda operações (que alterem o estado como registers e posts) num ficheiro json
- Sempre que uma nova operação chega ao servidor, esta é validada (os argumentos são verificados); registada no ficheiro; executada e uma resposta é enviada ao cliente
- Para garantir a integridade do ficheiro e evitar corrupção, por exemplo nos casos em que ocorre um crash enquanto o ficheiro é gravado, o sistema efetua primeiro o *save* num ficheiro alternativo (o *swap file*) e depois substitui atomicamente o *save* pelo *swap file*

Concorrência

O servidor consegue efetuar o seu serviço corretamente mesmo com pedidos de vários clientes em simultâneo. Foram efetuados 500 pedidos em simultâneo e verificado que o estado era correto. A única limitação encontra-se na secção 4.

4. Análise de ameaças e mecanismos de proteção

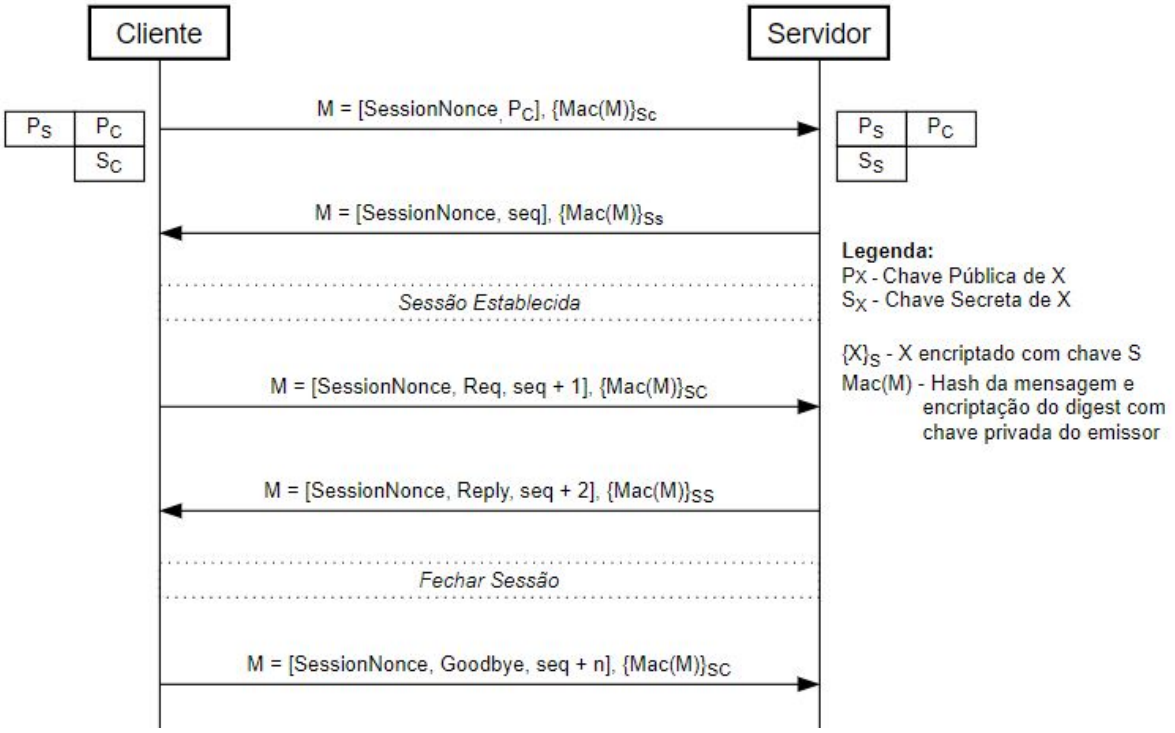

Figura 2: Protocolo

Ataques contemplados:

Drop e Rejeição

Frescura é garantida com sequence numbers. Um atacante que faça drop de uma mensagem pode forçar dessincronização. O servidor vai observar dessincronia nos sequence numbers recebidos e lançar uma excepção adequada. Se o cliente obtiver esta excepção, inicia uma nova sessão. Caso o atacante faça drop da última mensagem, a sessão irá expirar e ser apagada.

Manipulação e alteração

O atacante não consegue gerar MACs válidos pelo que uma mensagem alterada vai ser rejeitada.

Duplicação

Frescura é garantida com sequence numbers.

DOS

Um atacante pode tentar gerar sessões para gastar a memória do servidor. Para prevenir isto, uma nova sessão tem um tempo de vida mais pequeno que uma sessão já existente, sendo estendido quando um pedido é validado para essa sessão.

Detalhes de implementação

O cliente não pode fazer pedidos concorrentes durante uma sessão, pois pode causar uma dessincronização nos sequence numbers.

Isto advém de que o pedido que é processado primeiro depende do escalonamento das threads no servidor. Uma possível solução seria um mecanismo *sliding window*, quenão foi implementado para manter a simplicidade. O front-end do cliente só efectua pedidos bloqueantes, nunca assíncronos, e um atacante não consegue forçar a dessincronização pois não consegue gerar pedidos válidos em nome de outro cliente.

Para garantir que cada referência é única mesmo para posts iguais cada post tem um número de sequência único atribuído pelo servidor.