Ciências ULisboa | Informática

di.ciencias.ulisboa.pt
/difcul

Segurança e Confiabilidade

2020/2021

Trabalho 1 – 2ª fase

1 Objetivos

A parte prática da disciplina de Segurança e Confiabilidade pretende familiarizar os alunos com alguns dos problemas envolvidos na programação de aplicações distribuídas seguras, nomeadamente a gestão de chaves criptográficas, a geração de sínteses seguras, cifras e assinaturas digitais, e a utilização de canais seguros à base do protocolo TLS. O primeiro trabalho a desenvolver na disciplina será realizado utilizando a linguagem de programação Java e a API de segurança do Java, e é composto por duas fases.

A primeira fase do projeto teve como objetivo fundamental a construção de uma aplicação distribuída a ser executada numa *sandbox*, focando-se essencialmente nas funcionalidades da aplicação. O trabalho consistiu na concretização do sistema ***SeiTchiz*** (uma versão portuguesa do Instagram), que é um sistema do tipo cliente-servidor que permite que os utilizadores (clientes) utilizem um servidor central para partilhar fotografias e comunicar com outros utilizadores. O sistema suporta dois modos de funcionamento. No modo **mural** (*feed*), os utilizadores colocam fotografias no seu perfil público guardado no servidor e podem seguir quaisquer outros utilizadores, vendo no seu mural as fotografias que estes publicaram. O serviço também permite a colocação de *likes* nestas fotografias. No modo **conversação**, podem enviar mensagens para grupos privados de utilizadores e ler as mensagens enviadas para os grupos a que pertencem. Cada utilizador possui uma conta no servidor, pode seguir qualquer outro utilizador livremente, e pode pertencer a vários grupos de acesso privado.

Na segunda fase do projeto vamos manter as várias funcionalidades definidas na primeira fase e adicionaremos vários mecanismos de segurança no sistema.

2 Modelo de sistema e definições preliminares

A arquitetura do sistema segue o que foi definido na primeira fase do projeto, no entanto, agora **todas as comunicações serão feitas através de *sockets* seguros TLS com autenticação unilateral**.

A fim de simplificar o projeto, assumimos que **todos os clientes têm acesso a todas as chaves públicas do sistema** (i.e., do servidor e de todos os utilizadores). Na prática, esta assunção abstrai a existência de uma infraestrutura de chave pública no sistema.

As chaves privadas estarão armazenadas em ***keystores*** (uma por cada utilizador e uma para o servidor) protegidas por passwords. Adicionalmente, todas as chaves públicas devem estar disponíveis como ficheiros **cert** (certificados no formato X509 auto-assinados) numa pasta com o nome *PubKeys* disponível em todas as máquinas. Adicionalmente, o certificado de chave pública (auto-assinado) do servidor deve ser adicionado a uma *truststore* a ser usada por todos os clientes. Todos os pares de chaves serão gerados usando o algoritmo RSA de 2048 bits.

Cada grupo mantido pelo servidor usará uma **chave de grupo** simétrica AES para cifrar e decifrar mensagens trocadas nesse grupo. A cifra será fim-a-fim, i.e., o servidor não terá acesso ao conteúdo

das mensagens trocadas, significando que **ambas as operações de cifrar e decifrar são efetuadas pelo cliente**. Por exemplo, quando um utilizador envia uma mensagem para um grupo, esta será cifrada no cliente antes do seu envio. O servidor recebe a mensagem cifrada e armazena-a. Da mesma forma, quando um utilizador pede ao servidor as mensagens do grupo, este envia para a máquina cliente as mensagens cifradas (tal como estão armazenadas) e será o cliente que as decifra e as mostra ao utilizador.

Finalmente, para maximizar ainda mais a confiança no ambiente de execução, **o servidor deve armazenar a lista de utilizadores, a lista de seguidores de cada utilizador, e a associação entre utilizadores e grupos, em ficheiros cifrados**. Isto garante que ninguém além do servidor corretamente inicializado consegue ler esses ficheiros. A informação sobre *likes* e as próprias fotografias não precisam de ser cifradas. Contudo, **o servidor deve ser capaz de verificar se a integridade das fotografias armazenadas não foi comprometida**, antes de as enviar aos clientes.

3 Funcionalidades

O sistema tem os seguintes requisitos:

1. O servidor recebe na linha de comandos a seguinte informação:

   SeiTchizServer <port> <keystore> <keystore-password>

   - <port> identifica o porto (TCP) para aceitar ligações de clientes;
   - <keystore> que contém o par de chaves do servidor;
   - <keystore-password> é a password da *keystore*.

2. O cliente pode ser utilizado com as seguintes opções:

   SeiTchiz <serverAddress> <truststore> <keystore> <keystore-password> <clientID>

   Em que:

   - <serverAddress> identifica o servidor (*hostname* ou endereço IP e porto; por exemplo 1.2.3.4:5678);
   - <truststore> que contém o certificado de chave pública do servidor;
   - <keystore> que contém o par de chaves do clientID;
   - <keystore-password> é a password da *keystore*;
   - <clientID> identifica o utilizador local (cliente). Caso o utilizador não esteja registado no servidor, efetua o seu registo, ou seja, adiciona este *clientID* e a identificação do seu certificado ao ficheiro de utilizadores do servidor.

3.1 Estabelecimento do canal seguro

Dado que na primeira fase do trabalho a comunicação entre cliente e servidor era feita de forma insegura (canais em claro e sem autenticação do servidor), nesta segunda fase será necessário resolver este problema de segurança. Em concreto, será necessário garantir a **autenticidade do servidor** (um atacante não deve ser capaz de fingir ser o servidor e assim obter os dados do utilizador) e a **confidencialidade** da comunicação entre cliente e servidor (um atacante não deve ser capaz de escutar a comunicação). Para este efeito, devem-se usar **canais seguros** (protocolo

TLS/SSL) e a verificação da identidade do servidor à base de criptografia assimétrica (fornecida pelo protocolo TLS). Nesta fase do trabalho é então necessário:

- Utilizar ligações TLS: deve-se substituir a ligação TCP por uma ligação TLS/SSL, uma vez que o protocolo TLS vai verificar a autenticidade do servidor e garantir a integridade e confidencialidade de toda a comunicação.
- Configurar as chaves necessárias: a utilização do protocolo TLS exige configurar as chaves tanto no cliente (*truststore* com o certificado auto-assinado do servidor) como no servidor (*keystore* com a sua chave privada e certificado da sua chave pública).

3.2 Autenticação do cliente

Diferentemente da primeira fase do projeto, **não vamos utilizar autenticação via password de utilizador**, mas sim baseada em criptografia assimétrica. Desta forma, a autenticação se dará em dois passos (após a ligação TLS ser estabelecida):

1) Cliente envia clientID ao servidor pedindo para se autenticar. O servidor responde com um *nonce* aleatório de 8 bytes (por exemplo, um *long*) e, caso o utilizador não esteja registado, uma *flag* a identificar que o utilizador é desconhecido.
2) Duas possibilidades:
   a) Se clientD ainda não tiver sido registado (i.e., é desconhecido), o cliente efetua o registo do utilizador enviando ao servidor o *nonce* recebido, a sua assinatura deste gerada com a chave privada do utilizador, e o certificado com a chave pública correspondente. O servidor verifica se o *nonce* recebido foi o gerado por ele no passo 1) e se a assinatura pode ser corretamente verificada com a chave pública enviada (provando assim que o cliente tem acesso à chave privada daquele utilizador, i.e., que é quem diz ser). Se esses testes forem bem-sucedidos, o servidor completa o registo, armazenando o par <clientID, chave pública> no ficheiro *users.txt* (que agora deve ser cifrado pelo servidor), onde o parâmetro "chave pública" é o nome do ficheiro que contém o certificado do utilizador. Após a conclusão do registo, o servidor envia ao cliente uma mensagem a informar que o utilizador está registado e autenticado. Caso o *nonce* ou a assinatura sejam inválidos, é devolvida uma mensagem de erro ao cliente informando que o registo e a autenticação não foram bem-sucedidos.
   b) Se clientID estiver registado no servidor, o cliente assina o *nonce* recebido utilizando a chave privada do utilizador, e envia esta assinatura ao servidor. O login é bem-sucedido se e somente se o servidor verificar a assinatura do *nonce* (que deve ser o mesmo gerado e enviado no passo 1) usando a chave pública associada ao clientID. Caso não se verifique a assinatura, é enviada uma mensagem de erro ao cliente informando que a autenticação não foi bem-sucedida.

3.3 Comandos

Uma vez realizada a autenticação com sucesso, a aplicação deverá apresentar um menu disponibilizando os mesmos comandos definidos na primeira fase do projeto, mas que agora devem ser concretizados de modo a satisfazer certos requisitos de segurança:

- **f**ollow <userID> – adiciona o cliente (clientID) à lista de seguidores do utilizador *userID*. Se o cliente já for seguidor de *userID* ou se o utilizador *userID* não existir deve ser assinalado um

erro.

- **u**nfollow <userID> - remove o cliente (clientID) da lista de seguidores do utilizador *userID*. Se o cliente não for seguidor de *userID* ou se o utilizador *userID* não existir deve ser assinalado um erro.
- **v**iewfollowers – obtém a lista de seguidores do cliente ou assinala um erro caso o cliente não tenha seguidores.
- **p**ost <photo> – envia uma fotografia (*photo*) para o perfil do cliente armazenado no servidor. Assume-se que o conteúdo do ficheiro enviado é válido, ou seja, é uma fotografia (não é necessário fazer nenhuma verificação). As fotografias devem ser armazenadas em claro (presume-se que são públicas). Contudo, **a integridade das fotografias tem de ser garantida**, pelo que o servidor deve gerar e guardar uma síntese segura de cada fotografia, que possa ser usada após leitura da fotografia do disco para verificar se esta não foi alterada.
- **w**all <nPhotos> - recebe as *nPhotos* fotografias mais recentes que se encontram nos perfis dos utilizadores seguidos, bem como o número de *likes* de cada fotografia (mostrando tudo no mural). Se existirem menos de *nPhotos* disponíveis, recebe as que existirem (ou assinala um erro se não existir nenhuma).
- **l**ike <photoID> – coloca um *like* na fotografia *photoID*. Todas as fotografias têm um identificador único atribuído pelo servidor. Os clientes obtêm os identificadores das fotografias através do comando **w**all. Se a fotografia não existir deve ser assinalado um erro.
- **n**ewgroup <groupID> – cria um grupo privado, cujo dono (*owner*) será o utilizador que o criou. Ao criar um grupo, o cliente cria uma **chave de grupo** simétrica (com o identificador 0, ver abaixo) e cifra-a com a chave pública do dono do grupo. Esta chave cifrada será armazenada no servidor. Se o grupo já existir assinala um erro.
- **a**ddu <userID> <groupID> – adiciona o utilizador *userID* como membro do grupo indicado. Apenas os donos dos grupos podem adicionar utilizadores aos seus grupos, pelo que deverá ser assinalado um erro caso o cliente não seja dono do grupo. Esta operação poderia ser feita associando o *userID* ao grupo e cifrando a chave de grupo com a chave pública do *userID* e enviando-a para o servidor. No entanto, **queremos garantir que o novo membro do grupo seja capaz de ler apenas as mensagens publicadas a partir do momento em que foi adicionado ao grupo**. Isso é feito através da troca da chave de grupo a ser usada dali em diante. Assim, o dono do grupo terá de (1) criar uma nova chave de grupo (cujo identificador será o da última chave incrementado de 1), (2) cifrar essa chave de grupo com a chave pública de cada um dos membros do grupo (incluindo o novo membro) e (3) enviar para o servidor uma lista com a nova chave de grupo cifrada com as chaves públicas dos membros do grupo. Cada elemento da lista é um par <userID, userID-NewGroupKey>, onde a *userID-NewGroupKey* corresponde à nova chave cifrada (e seu identificador) com a chave pública do *userID*. As novas publicações no grupo serão cifradas com a nova chave do grupo, sendo necessário garantir que as mensagens anteriores conseguem ser decifradas pelos utilizadores (ver comandos collect e history). Para isso, o servidor mantém, para cada grupo, uma lista dos seus membros no formato <userID, userID-GroupKeys>, onde a *userID-GroupKeys* é o ficheiro contendo as chaves de grupo (e seus identificadores) usadas enquanto esse utilizador estava no grupo, cifradas com a chave pública do *userID*. Se *userID* já pertencer ao grupo ou se o grupo não existir deve ser assinalado um erro.

- **r**emoveu <userID> <groupID> – remove o utilizador *userID* do grupo indicado. Se *userID* não pertencer ao grupo ou o grupo não existir deve ser assinalado um erro. Apenas os donos dos grupos podem remover utilizadores dos seus grupos, pelo que deverá ser assinalado um erro caso o cliente não seja dono do grupo. **Ao remover um utilizador de um grupo, é necessário garantir que esse utilizador não seja capaz de ver mensagens publicadas após a sua remoção**. Isso é garantido trocando a chave de grupo a ser usada a partir da próxima publicação – conforme descrito acima – e não dando acesso à nova chave de grupo ao membro excluído.
- **g**info [groupID] – se *groupID* não for especificado, mostra uma lista dos grupos de que o cliente é dono, e uma lista dos grupos a que pertence. Caso não seja dono de nenhum grupo ou não seja membro de nenhum grupo, esses factos deverão ser assinalados. Se for especificado o *groupID,* mostra o dono do grupo e uma lista dos membros do grupo. Se o *groupID* especificado não existir ou se o cliente não for dono nem membro do grupo, deve ser assinalado um erro.
- **m**sg <groupID> <msg> – envia uma mensagem (*msg*) para o grupo *groupID*, que ficará guardada numa caixa de mensagens do grupo, no servidor. **A mensagem deve ser enviada ao servidor cifrada com a chave do grupo mais atual**. **O servidor terá de guardar informação necessária que associe à mensagem a chave de grupo que a cifrou, i.e., o identificador da chave**. A mensagem ficará acessível aos membros do grupo através do comando **c**ollect. Se o grupo não existir ou o cliente não pertencer ao grupo deve ser assinalado um erro.
- **c**ollect <groupID> – recebe todas as mensagens que tenham sigo enviadas para o grupo *groupID* e que o cliente ainda não tenha recebido anteriormente. Note que as **mensagens poderão vir cifradas com diferentes chaves de grupo e devem ser decifradas usando a chave de grupo apropriada** (que está cifrada com a chave pública de cada utilizador do grupo). Se não existir nenhuma nova mensagem, esse facto deverá ser assinalado. Os utilizadores apenas têm acesso às mensagens enviadas depois da sua entrada no grupo. Se o grupo não existir ou o cliente não pertencer ao grupo deve ser assinalado um erro.
- **h**istory <groupID> – mostra o histórico das mensagens do grupo indicado que o cliente já leu anteriormente. Note que as **mensagens poderão vir cifradas com diferentes chaves de grupo e devem ser decifradas usando a chave de grupo apropriada** (que está cifrada com a chave pública de cada utilizador do grupo). Se o grupo não existir ou o cliente não pertencer ao grupo deve ser assinalado um erro.

Como definido na primeira fase do projeto, o **servidor deve correr numa *sandbox*** que limite o seu acesso à rede e ao sistema de ficheiros, adaptadas para usar as funcionalidades de segurança. O **cliente também deve correr numa *sandbox***. Para além disso, o grupo pode adicionar outras políticas que julgue necessárias para o correto funcionamento do sistema.

4 Entrega

4.1 Trabalho

Dia **5 de abril**, até às 23:55 horas. O código desta segunda (e última) fase do trabalho deve ser entregue de acordo com as seguintes regras:

- A segunda fase do projeto deve ser realizada mantendo os grupos da primeira fase.

- Para entregar o trabalho, é necessário criar um **ficheiro zip** com o nome **SegC-grupoXX-proj1-2.zip**, onde XX é o número do grupo, contendo:
  - ✓ o código do trabalho;
  - ✓ conjunto de chaves, *keystores*, certificados (ficheiros de formato **cert**) e *truststore*;
  - ✓ ficheiros de permissões (cliente e servidor);
  - ✓ os ficheiros jar (cliente e servidor) para execução do projeto.
- O ficheiro zip é submetido através da atividade disponibilizada para esse efeito na página da disciplina no Moodle. Apenas um dos elementos do grupo deve submeter. Se existirem várias submissões do mesmo grupo, será considerada a mais recente.

4.2 Relatório

Dia **7 de abril**, até às 23:55 horas. O relatório do trabalho deve ser entregue em formato **PDF**, com um máximo de 4 páginas, e deve descrever ou indicar:

- Como compilar o projeto;
- Como executar o projeto com os ficheiros de permissões (cliente e servidor);
- As passwords das *keystores* dos utilizadores correspondentes.
- Decisões de desenho das soluções implementadas, nomeadamente em termos de arquitetura de software, de segurança, de desempenho e de funcionalidade;
- Limitações do trabalho.
- O ficheiro PDF é submetido através da atividade disponibilizada para esse efeito na página da disciplina no Moodle. Apenas um dos elementos do grupo deve submeter. Se existirem várias submissões do mesmo grupo, será considerada a mais recente.

4.3 Auto-avaliação

Dia **8 de abril**, até às 23:55 horas. Cada aluno preenche no Moodle um formulário de auto-avaliação das contribuições individuais de cada elemento do grupo, considerando as duas fases do trabalho. Por exemplo, se todos os elementos colaboraram de forma idêntica, bastará que todos indiquem que cada um contribuiu 33%. Aplicam-se as seguintes regras e penalizações:

- Alunos que não preencham o formulário sofrem uma penalização na nota de 10%. Sujeitam-se ainda à avaliação que os colegas de grupo indicarem.
- Se nenhum elemento do grupo preencher o formulário, todos terão uma penalização na nota de 10%. As notas individuais serão idênticas.
- Nos casos em que se verifiquem assimetrias significativas entre as apreciações feitas por cada um dos elementos do grupo, o corpo docente poderá convocar os elementos do grupo para uma discussão sobre o trabalho.

4.4 Demonstrações e discussões

Não estão previstas discussões com todos os grupos. Contudo, qualquer grupo poderá ser chamado para demonstrar e discutir o seu trabalho, sendo esta discussão a realizar no fim do semestre.

Não serão aceites trabalhos enviados por email nem por qualquer outro meio não definido nesta secção. Se não se verificar algum destes requisitos o trabalho é considerado não entregue.