Ciências ULisboa | Informática

di.ciencias.ulisboa.pt

/difcul

Segurança e Confiabilidade

2021/2022

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 tem como objetivo fundamental a construção de uma aplicação distribuída, focando-se essencialmente nas funcionalidades da aplicação. O trabalho consiste na concretização do sistema ***Trokos,*** que é um sistema do tipo cliente-servidor que oferece um serviço semelhante ao do MB WAY. Existe uma **aplicação servidor** que tem acesso às contas bancárias dos utilizadores registados, e que oferece diversas funcionalidades relacionadas com a movimentação dessas contas (transferências, pagamentos, pedidos de pagamento, etc.). Os utilizadores registados devem usar uma **aplicação cliente** para interagirem com o servidor e usar essas funcionalidades.

Para facilitar o desenvolvimento e teste do sistema *Trokos*, considera-se que é o próprio servidor que mantém informação sobre as contas bancárias dos clientes e o respetivo saldo, não sendo assim necessário implementar interfaces para acesso aos servidores dos bancos.

Na primeira fase do trabalho foram concretizadas as funcionalidades oferecidas pelo serviço. Nesta segunda fase do projeto serão considerados requisitos de segurança, para que as interações e o sistema no seu todo sejam seguros. As várias funcionalidades definidas na primeira fase serão mantidas sem qualquer alteração, embora a sua concretização tenha em alguns casos de ser adaptada para satisfazer os requisitos de segurança.

2 Modelo de sistema e definições preliminares

A arquitetura do sistema segue o que foi definido na primeira fase do projeto. Contudo, agora **todas as comunicações serão feitas através de *sockets* seguros TLS com autenticação unilateral**.

As chaves privadas estarão armazenadas em ***keystores*** (uma por cada utilizador e uma para o servidor) protegidas por passwords. 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.

Finalmente, para maximizar ainda mais a confiança no ambiente de execução, **o servidor deve armazenar todos os ficheiros cifrados. A única exceção são os ficheiros com a blockchain das transações**, que terão apenas a integridade protegida (ver a seguir). Isto garante que ninguém, além do servidor, corretamente inicializado consegue ler esses ficheiros. A chave a ser usada na cifra dos ficheiros deve ser baseada numa password apresentada na inicialização do servidor, usando, portanto, o algoritmo PBE (Password Based Encryption) com AES de 128 bits.

3 Utilização do Sistema Seguro

A execução das aplicações servidor e cliente deve ser feita da seguinte forma:

1. *TrokosServer* <port> <password-cifra> <keystore> <password-keystore>
    - <port> identifica o porto (TCP) para aceitar ligações de clientes. Por omissão o servidor deve usar o porto 45678;
    - <password-cifra> é a password a ser usada para gerar a chave simétrica que cifra os ficheiros da aplicação;
    - <keystore> que contém o par de chaves do servidor;
    - <password-keystore> é a password da *keystore*.
2. *Trokos* <serverAddress> <truststore> <keystore> <password-keystore> <userID>
    - <serverAddress> identifica o servidor. O formato de serverAddress é o seguinte: <IP/*hostname*>[:Port]. O endereço IP ou *hostname* do servidor são obrigatórios e o porto é opcional. Por omissão, o cliente deve ligar-se ao porto 45678 do servidor;
    - <truststore> que contém o certificado de chave pública do servidor;
    - <keystore> que contém o par de chaves do userID;
    - <password-keystore> é a password da *keystore*;
    - <userID> identifica o utilizador local.

4 Adicionar Segurança ao Sistema

4.1 Canais seguros TLS para comunicação segura e autenticação de servidores

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).

4.2 Autenticação de utilizadores

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

1) Cliente envia *userID* 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 *userID* ainda não tiver sido registado (i.e., é desconhecido), o cliente efetua o registo do utilizador enviando ao servidor o *nonce* recebido, a assinatura deste gerada com a sua chave privada, 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 *userID* estiver registado no servidor, o cliente assina o *nonce* recebido com a sua chave privada, e envia esta assinatura ao servidor. O login é bem-sucedido apenas 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 *userID*. Caso não se verifique a assinatura, é enviada uma mensagem de erro ao cliente informando que a autenticação não foi bem-sucedida.

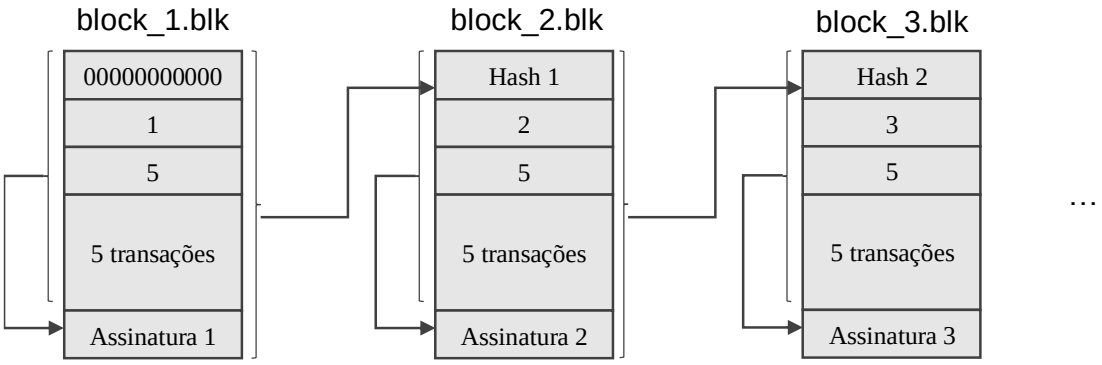
4.3 Criação de um log seguro para transações

O servidor deve manter um log incremental seguro das transações executadas, a nossa *blockchain*. Para isso, duas coisas são necessárias:

1) No caso das funcionalidades que implicam a movimentação de valores (*makepayment*, *payrequest e confirmQRcode*), as transações têm de ser assinadas pelos clientes antes de serem executadas pelo servidor. Considera-se que uma transação é constituída pela identificação de um destinatário e pelo valor do pagamento, sendo a transação validada pela assinatura de quem paga. Estas transações devem ser incluídas na *blockchain* na ordem em que são processadas, juntamente com a respetiva assinatura.
2) A cada 5 transações adicionadas ao log, este deve ser *lacrado* como um bloco. Neste momento, o servidor deve assiná-lo e depois calcular o seu *hash* SHA256 (já incluindo os bytes da assinatura). O próximo bloco deve começar com um cabeçalho que contem o *hash* do bloco anterior (ou uma cadeia de 32 bytes a zero, caso seja o primeiro bloco), o número deste bloco (um long), e a quantidade de transações dentro do bloco (um long).

Cada bloco deve ser armazenado num ficheiro separado com o nome *block_<numero>.blk*.

O objetivo de criar um log com esse formato é permitir que a informação sobre as transações seja exportada para outros sistemas (ex: o sistema da banca), de forma incremental (bloco a bloco), sem perder a segurança. Note que o log gerado será constituído por um conjunto de ficheiros como o ilustrado a seguir.



5 Funcionalidades

Os comandos suportados pelo cliente *Trokos* devem ser os mesmos definidos na 1ª fase do projeto.

Contudo, a concretização das funcionalidades *makepayment*, *payrequest* e *confirmQRcode* deverá ser adaptada para que o cliente envie ao servidor a assinatura da transação. Assim, temos que:

- Na operação *makepayment*, o cliente já tem o valor e o destinatário do pagamento (fornecidos pelo utilizador), tendo de os assinar e de adicionar a assinatura ao pedido enviado ao servidor.
- Na operação *payrequest*, o cliente pode obter o valor e o destinatário do pagamento através da operação *viewrequests*. Assina esta informação e envia a assinatura ao servidor, juntamente com a identificação do pedido de pagamento (o servidor já tem os dados que forram assinados).
- Na operação *confirmQRcode*, depois do cliente enviar a operação ao servidor, terá de ficar à espera que o servidor envie de volta o valor e o destinatário do pagamento associados ao QRcode, para que este possa assinar a transação e enviar a assinatura. Assim, neste caso haverá uma interação adicional entre o cliente e o servidor, para completar a transação.

6 Entrega

6.1 Trabalho

Dia **29 de abril**, até às 23:59 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-fase2.zip**, onde **XX** é o número do grupo, contendo:
  - ✓ o código do trabalho;
  - ✓ conjunto de chaves, *keystores*, certificados (ficheiros de formato **cert**) e *truststore*;
  - ✓ os ficheiros jar (cliente e servidor) para execução do projeto;
  - ✓ um readme (txt) indicando **como compilar** e **como executar** o projeto, indicando também as limitações do trabalho.
- 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.
- Não serão aceites trabalhos enviados por email. Se não se verificar algum destes requisitos o trabalho é considerado não entregue.

6.2 Autoavaliação de contribuições

Dia **1 de maio**, até às 23:59 horas. Cada aluno preenche no Moodle um formulário de autoavaliação das contribuições individuais de cada elemento do grupo nesta 2ª fase. 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%.
- Nos casos de verificarem assimetrias entre as contribuições de cada elemento do grupo ou entre as autoavaliações de cada elemento do grupo, estas serão analisadas durante a discussão do trabalho e poderão levar à atribuição de notas individuais diferentes para cada elemento do grupo.