Ciências ULisboa | Informática

di.ciencias.ulisboa.pt

/difcul

Segurança e Confiabilidade

2022/2023

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 ***Tintolmarket,*** que é um sistema do tipo cliente-servidor que oferece um serviço semelhante ao do *Vivino,* mas permitindo a compra e venda de vinhos por parte dos utilizadores do sistema. Existe uma **aplicação servidor** que mantém informação sobre os utilizadores registados e sobre vinhos listados, o seu valor, classificação e quantidade disponibilizada por cada utilizador. Os utilizadores podem adicionar vinhos, indicar quantidades que dispõem, classificar cada vinho, e ainda enviar mensagens privadas a outros utilizadores. Os utilizadores registados devem usar uma **aplicação cliente** para interagirem com o servidor e usar essas funcionalidades.

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. Para além disso, e dado que todos os utilizadores terão necessidade de usar as chaves públicas uns dos outros e do servidor, os certificados de chave pública (auto-assinados) do servidor e dos clientes devem ser adicionados a uma ***truststore*** a ser usada por todos os clientes. Todos os pares de chaves serão gerados com o algoritmo RSA de 2048 bits.

Para maximizar ainda mais a confiança no ambiente de execução, **o servidor tem de cifrar o ficheiro de utilizadores.** Isto garante que ninguém, além do servidor, consegue ler este ficheiro. A chave a ser usada na cifra do ficheiro deve ser baseada numa password apresentada na inicialização do servidor, usando, portanto, o algoritmo PBE (Password Based Encryption) com AES de 128 bits. Quanto aos restantes ficheiros mantidos pelo servidor, será necessário assegurar que não foram corrompidos, **verificando sempre a sua integridade** antes de usar a informação neles contida.

Será ainda necessário **criar um log seguro para registar todas as transações** realizadas no sistema, que tomará a forma de uma *blockchain*.

Finalmente, será necessário **garantir confidencialidade end-to-end** para as mensagens trocadas entre clientes.

3 Utilização do Sistema Seguro

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

1. *TintolmarketServer* <port> <password-cifra> <keystore> <password-keystore>
    - <port> identifica o porto (TCP) para aceitar ligações de clientes. Por omissão o servidor deve usar o porto 12345;
    - <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. *Tintolmarket* <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 12345 do servidor;
    - <truststore> que contém os certificados de chave pública do servidor e de outros clientes;
    - <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 tem de ir construindo e tem de manter um log incremental seguro das transações executadas, que será construído como uma *blockchain*. Neste projeto, consideram-se como transações as operações *sell* e *buy* realizadas pelos clientes do *TintolmarkerServer*.

A operação *sell* ocorre quando um utilizador coloca novas garrafas no mercado. A operação pode ser vista como correspondendo à criação de um (ou vários) NFT (*Non-Fungible Token*), ou seja, uma ou várias garrafas de vinho. A **transação** a ser guardada no log terá o seguinte conteúdo: a **identificação do vinho**, o **número de unidades criadas**, o **valor de cada unidade**, e a **identificação do seu dono**, ou seja, do próprio utilizador que realizou a operação. Toda esta informação tem também de ser assinada pelo utilizador que realizou a operação.

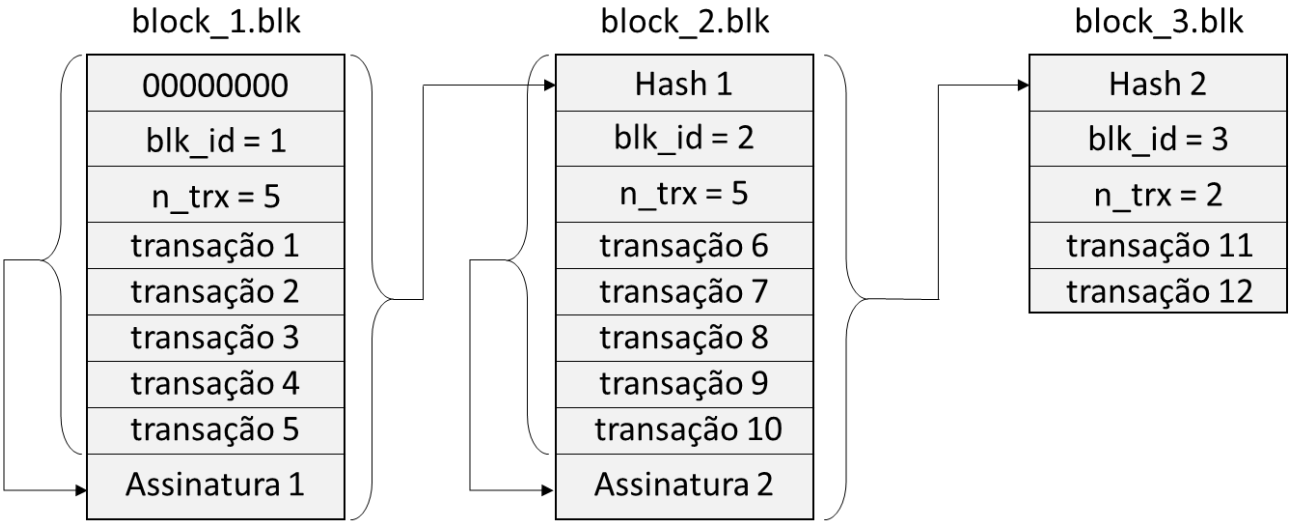
A operação *buy* permite comprar uma ou várias garrafas em troco de um determinado valor. Neste caso existe a operação corresponde à troca de um NFT (garrafa de vinho) por um determinado valor de uma qualquer moeda, ou seja, de um FT (*Fungible Token*). Se o comprador tiver saldo suficiente, a transação pode ser efetuada e deve ficar registada com o seguinte conteúdo: a **identificação do vinho**, o **número de unidades vendidas**, o **valor de cada unidade**, e a **identificação do comprador**, ou seja, do novo dono. A informação também tem de ser assinada pelo comprador (que assim confirma a transferência do dinheiro para o vendedor).

Em suma, para que estas operações sejam executadas pelo servidor, o cliente tem de enviar a

informação assinada, e a assinatura tem de ser verificada pelo servidor (para além de outras verificações). Cada uma destas operações corresponde a uma transação que tem de ser registada no log (blockchain) pela ordem em que foi realizada pelo servidor.

Para além disso, 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* como o ilustrado a seguir.



Finalmente, sempre que o servidor é inicializado deve verificar que a integridade da *blockchain* se mantém, o que implica a verificação das assinaturas e do *hash* de cada bloco. Dado que o último bloco pode não estar completo (como na figura), não é feita nenhuma verificação da integridade deste bloco incompleto. As novas transações serão adicionadas na continuação do último bloco que existir.

4.4 Confidencialidade fim-a-fim para as mensagens

Na 1ª fase do projeto as mensagens enviadas de um utilizador para outro podiam ser lidas pelo servidor e eram guardadas em claro num ficheiro. Pretende-se agora nesta fase que seja assegurada confidencialidade fim-a-fim, ou seja, que apenas o utilizador para quem a mensagem é enviada a possa ler. Tal será feito utilizando criptografia assimétrica. Os utilizadores terão acesso às chaves públicas de todos os restantes utilizadores, e deverão cifrar as mensagens com a chave pública do destinatário.

5 Funcionalidades

Deve ser acrescentada a seguinte nova funcionalidade ao cliente *Tintolmarket*:

- **l**ist - obtém a lista de todas as transações que já foram efetuadas e que se encontram armazenadas na blockchain.

6 Entrega

6.1 Trabalho

Dia **28 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 **30 de abril**, até às 23:59 horas. Cada aluno preenche no Moodle um formulário de autoavaliação das contribuições individuais de cada elemento do grupo na 2º fase do projeto. 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%.
- Caso existam 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.