Voting Protocol

Riei Joaquim Matos Rodrigues - rjmr
Victor Hugo Meirelles Silva - vhms

1. Dependências:

- Bibliotecas específicas:
  - cryptography: biblioteca de criptografia que inclui receitas de alto nível e interfaces de baixo nível para algoritmos criptográficos comuns. Utilizamos elas para criptografia RSA de pacotes e para autenticação de pacotes, vindos do servidor no cliente.
  - pycryptodome: biblioteca de criptografia de primitivas criptográficas de baixo nível. Utilizamos ela para criptografia AES de pacotes de descoberta de servidores por clientes.

- Bibliotecas gerais:
  - threading: Utilizada no servidor para poder atender a múltiplas conexões tanto TCP quanto UDP.
  - dataclasses: Utilizada para Criar as entidades, objetos que representam os pacotes de comunicação estabelecidos no protocolo.
  - datetime: Utilizada para gerenciar o tempo e verificar se uma sessão de votação está aberta ou concluída.
  - socket: Utilizada para para fazer e manipular as conexões TCP e UDP tanto da parte do cliente quanto do servidor.
  - json: Utilizada para serializar as entidades em string para serem criptografados e mandado pelo socket em bytes, além de ser usada para remontar a entidade a partir de uma string. Também é utilizada para armazenar as sessões existentes num arquivo que pode ser consultado e editado pelo servidor.
  - secrets: Utilizado para gerar o token de autenticação de login para validar as requisições de operações do usuário.
  - setuptools: Utilizado para Criar o pacote VotingProtocol e instalar as dependências necessárias para a seu funcionamento.

Para instalar as dependências use o comando ‘pip install -e .’ no terminal na pasta raiz do projeto.

2. Principais Componentes:

- **VotingServer:** Método principal para a execução da rotina do servidor, no protocolo de votação, recebe como parâmetro:
  - signaturePrivateKey: bytes da chave RSA privada para assinar as mensagens de setup enviadas clientes para que eles possam autenticar a origem dos parâmetros enviados.

    - userAccounts: dicionário de dados no formato ‘email’:’senha’, tanto senha quanto email em formato string, esse banco de dados servirá para o servidor verificar clientes válidos para votação no processo de login.
    - udpPort: um número inteiro para a porta UDP em que será esperado broadcast de descoberta de clientes e que será enviada o pacote de setup inicial do cliente.
    - tcpPort: um número inteiro para a porta TCP que o servidor utilizará para comunicação principal com o cliente, para fazer login, requisições e votações.
    - tcpCommCapabilities: número inteiro que define a capacidade de atendimentos simultâneos que o servidor poderá fazer.

- A rotina é executada em duas threads, uma processando requisições UDP e outra com requisições TCP. Cada cliente começa se comunicando com UDP e depois de autenticar o servidor e seus parâmetros passa a usar a conexão TCP para fazer suas requisições.

- **VotingClient:** Método principal para a execução da rotina de um cliente, no protocolo de votação, recebe como parâmetro:
    - signaturePublicKey: bytes da chave RSA pública para autenticação das mensagens de setup vindas do servidor.
    - udpPort: um número inteiro para a porta UDP que será utilizada para a descoberta e setup inicial com o servidor.
    - TcpPort: um número inteiro para a porta TCP que será utilizada para comunicação principal com o servidor, para fazer login, requisições e votações.
- Inicialmente o cliente busca localizar servidores na mesma rede fazendo broadcasts UDP de pacotes ‘Hello Servers’ criptografados com uma chave AES comum a clientes e servidores, esperando que o servidor responda com um pacote ‘Hello Client’ com assinatura válida, sendo autenticada a origem o parâmetro de IP do servidor será usado para estabelecer a conexão TCP com o servidor e a ServerPublicKey recebida passará a ser usada pelo cliente. Estabelecendo a conexão TCP o cliente passa a enviar seus dados de login e sua publicKey para o servidor, utilizando uma criptografia RSA desses dados. Tendo obtido sucesso no login o cliente recebe um token que autentica seu login nas requisições seguintes.

- **Communication:** Responsável pela implementação do protocolo de comunicação da votação.
    - **setupPorts:** Função comum ao servidor e ao cliente que recebe valores de porta UDP e TCP como parâmetros e os definem para serem usados nos sockets.

- **setServerCapabilities:** Função do servidor, que recebe como parâmetro a quantidade de conexões simultâneas que serão suportadas pelo servidor e a define para ser utilizada no seu socket.

- **setSignature:** Função comum ao servidor e ao cliente, que recebe e passa os parâmetros, na chamada para o método ‘setSignatureKey’ do objeto ProcessPacket da comunicação.

- **startUDPSocket:** Função comum ao servidor e ao cliente, que recebe uma flag como parâmetro, podendo ser ‘CLIENT’ ou ‘SERVER’, baseada no seu valor a função configura e retorna um objeto Socket configurado de acordo, para o funcionamento do protocolo.

- **UDPServerRunner:** Função blocante do servidor, com a rotina de receber pacotes broadcast com 'Hello Servers' criptografados com uma chave AES conhecida por clientes e servidores do protocolo e responder ao remetente com um pacote 'Hello Client' assinado com uma chave RSA previamente definida, para garantir autenticidade da origem do IP e da publicKey enviadas.

- **TCPServerRunner:** Função blocante do servidor, que estabelece as conexões TCPs com os clientes, quando um cliente se conecta ele destina uma thread de execução para processar e responder as requisições desse cliente, comunicação baseada na chaves RSA trocadas na fase UDP do protocolo.

- **UDPDiscoverValidsServers:** Função do cliente, que utilizando UDP envia broadcasts com 'Hello Servers' periodicamente na rede e recebe pacotes 'Hello Client' assinados em uma porta previamente definida. Essa loop se mantém até que seja recebido um pacote com assinatura válida, sendo retornado o objeto no payload.

- **setupRemoteServer:** Função que recebe como parâmetros o IP do servidor e sua publicKey. O IP é definido na comunicação para ser utilizado no socket TCP e a publicKey é passada para a chamada do método ‘setRemotePublicKey’ do objeto ProcessPacket da comunicação.

- **TCPClientRunner:** Função do cliente, que executa a rotina de votação, lendo os comandos do usuário no terminal, enviando as requisições para o servidor via TCP, comunicação criptografada baseada em RSA, recebendo as respostas, às processando e exibindo as informações correspondentes no terminal. A rotina se inicia com o

cliente no estado de login, onde os dados são recebidos do terminal e enviados para o servidor, caso o dados passados sejam de algum dos usuários previamente cadastrados, o lado do cliente entrará em um loop de receber requisições e dados do terminal, enviar para o servidor e exibir os dados da resposta recebida, seguindo a máquina de estados da documentação.

- **TCPClientConnection:** Função do servidor, que representa a conexão de um cliente, recebendo dele requisições e dados que serão processados, tanto para login do usuário quanto para armazenar votos depositados, sendo respondidos seguindo o fluxo de estados da documentação, comunicação utilizando criptografia RSA.

- **ProcessPackets:** Responsável por criar as chaves de criptografia locais e por gerenciar as chaves remotas recebidas, sendo responsável pelo encode e decode de pacotes. Por padrão no estado inicial a classe utiliza o AES como encrypter com uma chave comum conhecida pelos servidores e clientes.
  - **generateRSAKeyToSigners:** Método de setup usado para gerar a chave de assinaturas que certificam a autoridade do servidor para o cliente, a chave pública será passada para a função principal do cliente e a privada para a função principal do servidor. As chaves geradas serão impressas no terminal e retornada na chamada do método.

  - **setRemotePublicKey:** Setup para definir a publicKey do outro comunicante, requisito para podermos trocar o encryptor pelo RSA. Caso o parâmetro remotePublicKey recebido seja válido a classe passará a criptografar os pacotes com essa chave RSA.

  - **setSignatureKey:** Setup para definir a chave de assinatura que será utilizada para verificar autenticidade de pacotes, a chave passada como parâmetro será utilizada para verificar autenticidade caso o parâmetro operation seja 'VERIFY' ou para assinar caso operation seja 'SIGNER'.

  - **segmentPacket:** Fragmenta a string em uma lista com N blocos com a quantidade de bytes indicado. fundamental para utilizar criptografia RSA em pacotes grandes.
    - strObj: String a ser fragmentada.
    - blockSize: quantidade de bytes de dados por bloco.

    - **signPacket:** Assina um pacote em bytes recebidos com a chave RSA previamente setada, para garantir autenticidade do conteúdo. Retornando um pacote em bytes com a assinatura do conteúdo.

    - **verifySignPacket:** Verifica a assinatura de um pacote de bytes recebidos com a chave RSA previamente setada, para garantir autenticidade do conteúdo. Retornando o payload do pacote caso seja confirmado autenticado.

    - **encode:** Recebe um objeto DataClass, converte ele para String e criptografa usando o encrypter definido atualmente. Retornando um objeto bytes, pronto para ser transmitido pelo socket.

    - **decode:** Recebe um objeto em bytes e tenta converter ele para string e descriptografar usando o encrypter definido atualmente. O parâmetro typeObject define como será retornado o resultado do processo, caso seja None, será retornado um objeto dicionário com os campos e valores disponíveis, caso seja alguma entidade DataClass o resultado da descriptografia tentará ser formatado em um objeto dessa DataClass, em todos os casos voltando None em caso de erro.

- **Entities:** Objetos que representam os pacotes do protocolo no código do servidor e do cliente.