Projecto de Criptografia Aplicada (TP1)

Pretende-se construir um serviço de *Message Relay* que permita aos membros de uma organização trocarem mensagens com garantias de autenticidade. O serviço será suportado por um servidor responsável por manter o estado da aplicação e interagir com os utilizadores do sistema. Todo os intervenientes do sistema (servidor e utilizadores) serão identificados por um *identificador único* `<UID>` (mais detalhes sobre este identificador abaixo).

Note que o projecto consiste essencialmente em combinar diferentes componentes criptográficas desenvolvidas nas aulas práticas por forma a atingir os requisitos abaixo apresentados.

Descrição

O projecto consistirá no desenvolvimento de dois programas em *Python*:

- `msg_server.py` – servidor responsável por responder às solicitações dos utilizadores, e manter o estado da aplicação;
- `msg_client.py` – cliente executado por cada utilizador que permitirá aceder à funcionalidade oferecida pelo serviço.

Ambos os programas podem ser construídos adaptando os programas Client.py e Server.py, que por sua vez serviram de base para vários dos programas realizados nas aulas práticas.

Comandos da aplicação cliente

Os utilizadores do sistema irão interagir com recurso ao programa `msg_client.py` que aceitará os seguintes comandos (com correspondentes opções de linha de comando):

- `-user <FNAME>` – argumento opcional (que deverá surgir sempre em primeiro lugar) que especifica o ficheiro com dados do utilizador. Por omissão, será assumido que esse ficheiro é `userdata.p12`.
- `send <UID> <SUBJECT>` – envia uma mensagem com assunto `<SUBJECT>` destinada ao utilizador com identificador `<UID>`. O conteúdo da mensagem será lido do `stdin`, e o tamanho deve ser limitado a 1000 bytes.
- `askqueue` – solicita ao servidor que lhe envie a lista de mensagens **não lidas** da *queue* do utilizador. Para cada mensagem na *queue*, é devolvida uma linha contendo: `<NUM>:<SENDER>:<TIME>:<SUBJECT>`, onde `<NUM>` é o número de ordem da mensagem na *queue* e `<TIME>` um *timestamp* adicionado pelo servidor que regista a altura em que a mensagem foi recebida.
- `getmsg <NUM>` – solicita ao servidor o envio da mensagem da sua *queue* com número `<NUM>`. No caso de sucesso, a mensagem será impressa no `stdout`. Uma vez enviada, essa mensagem será marcada como lida, pelo que não será listada no próximo comando `askqueue` (mas pode voltar a ser pedida pelo cliente).
- `help` – imprime instruções de uso do programa.

No caso de erro na interpretação do comando (e.g. número incorrecto de argumentos), a aplicação deverá emitir o erro `MSG RELAY SERVICE: command error!` para o `stderr`, seguido das instruções de uso apresentadas pelo comando `help`.

Funcionalidade pretendida

1. Todas a comunicação entre os clientes e servidor devem ser protegidas contra acesso ilegítimo e/ou manipulação (incluindo de outros utilizadores do sistema).
2. Confia-se no servidor para efeitos da atribuição do *timestamp* associada à recepção das mensagems, e que este não compromete a confidencialidade das mensagens tratadas. No entanto não lhe deve ser permitida a manipulação do conteúdo ou destino dessas mensagens por ele tratadas.
3. Os clientes, ao receberem uma mensagem, devem poder verificar que a mensagem foi efectivamente enviada pelo `<SENDER>` especificado e a si dirigida. No caso de erro, deve ser enviado para para `stderr` a mensagem respectiva, nomeadamente:

- `MSG RELAY SERVICE: unknown message!` – no caso da mensagem não existir na *queue* do utilizador;
- `MSG RELAY SERVICE: verification error!` – no caso da verificação atrás mencionada falhar.

Identificação e credenciais dos utilizadores

1. A identificação dos intervenientes do sistema é representada na informação contida num certificado X509. Considera-se que o nome do campo `subject` desse certificado contém pelo menos os atributos: `PSEUDONYM`, que irá armazenar o `<UID>` do utilizador; `CN`, que contém um nome mais informativo (e.g. nome completo), e o atributo `OU` que irá conter sempre a string `SSI MSG RELAY SERVICE`. O `<UID>` do servidor será `MSG_SERVER`.
2. O ficheiro contendo os dados do cliente (por omissão, `userdata.p12`) irá consistir numa keystore `PKCS12` contendo o certificado do utilizador e a respectiva chave privada.
3. Todos os certificados considerados pelo sistema serão emitidos por uma EC dedicada (com `<UID> MSG_CA`). Essa entidade é confiada para efeitos de atribuição do acesso ao serviço e na consistencia das credenciais de acesso fornecidas (e.g. unicidade dos `<UID>` dos utilizados). Na directoria `projCA/certs` são disponibilizados certificados que podem ser usados pela aplicação:
   - `MSG_CA.crt` – certificado auto-assinado da EC do sistema;
   - `MSG_SERVER.p12`,`MSG_CLI1.p12`, `MSG_CLI2.p12` e `MSG_CLI3.p12` – *keystores* contendo os certificados e chave privadas do servidor e de três utilizadores. As *keystores* não dispõe de qualquer protecção[1]. Por conveniencia, as *keystores* contém ainda o certificado da EC do sistema.

Note que pode facilmente extrair o conteúdo das *keystores* recorrendo à classe PKCS12 da biblioteca `cryptography`. Por exemplo:

```
def get_userdata(p12_fname):
    with open(p12_fname, "rb") as f:
        p12 = f.read()
    password = None # p12 não está protegido...
    (private_key, user_cert, [ca_cert]) = pkcs12.load_key_and_certificates(p12, password)
    return (private_key, user_cert, ca_cert)
```

Possíveis valorizações

O projecto atrás apresentado oferece suficiente liberdade para permitir aos grupos interessados incluir melhorias tanto em termos de funcionalidade e/ou garantias de segurança; aspectos de implementação; etc. Os grupos são incentivados a identificar e implementar essas melhorias sendo que, se quebrar a compatibilidade com a versão original, o devem fazer mantendo disponíveis os programas originais. Podem ainda ser propostas e analisadas melhorias no relatório que acabem por não ser implementadas (e.g. por falta de tempo e/ou porque envolveriam alterações substanciais na arquitectura).

Algumas possibilidades:

- Suportar a componente de certificação (i.e. gerar os próprios certificados usados pela aplicação);
- Recorrer a JSON ou outro formato similar para estruturar as mensagens do protocolo de comunicação (eventualmente, recorrendo também a um *encoding* binário como BSON);
- Possibilitar o envio de mensagens com garantias de confidencialidade mesmo perante um servidor "curioso";
- Retirar a assumpção que o servidor é confiável para efeitos de atribuição do *timestamp* de recepção da mensagem;
- Contemplar a existência de recibos que atestem que uma mensagem foi submetida ao sistema;
- Sistema de *Log* que registe transações do servidor;
- etc.

[1] Prática seguida para facilitar o processo de desenvolvimento (mas claramente desaconselhada se se estivesse a falar de um produto final...).

Relatório

O projecto deve ser acompanhado por um relatório que descreva o sistemam, documente as opções tomadas, o trabalho realizado, e outra informação que considerem pertinente. Sugere-se que este relatório seja realizado directamente em `MarkDown` acessível a partir do ficheiro `README.md` colocado na directoria do projecto no repositório do grupo (`TPs/TP1/README.md`).