Semana 6

Comunicação segura entre Cliente/Servidor

As scripts Client.py e Server.py constituem uma implementação elementar de uma aplicação que permite a um número arbitrário de clientes comunicar com um servidor que escuta num dado port (e.g. 8443). Na sua forma actual, o servidor atribui um número de ordem a cada cliente e simplesmente faz o *dump* em `stdout` do texto enviado por esse cliente (prefixando cada linha com o respectivo número de ordem), e responde ao cliente com o texto convertido em maiúsculas. Quando um cliente fecha a ligação, o servidor assinala o facto (e.g. imprimindo [n], onde n é o número do cliente).

Exemplo da execução do servidor (que comunica com 3 clientes):

```
$ python3 Servidor.py
1 : daskj djdhs slfghfjs askj
1 : asdkdh fdhss
1 : sjd
2 : iidhs
2 : asdjhf sdga
2 : sadjjd d dhhsj
3 : djsh
1 : sh dh d   d
3 : jdhd kasjdh as
2 : dsaj dasjh
3 : asdj dhdhsjsh
[3]
2 : sjdh
1 : dhgd ss
[1]
2 : djdj
[2]
```

PROG: `Client_sec.py` e `Server_sec.py`

Pretende-se modificar os programas fornecidos por forma a protegeram a comunicação entre Cliente/Servidor. Para isso deverá fazer uso de uma cifra autenticada como as utilizadas no último guião (e.g. AES-GCM).

Observações:

- Nesta fase, deve desvalorizar as questões relativas à protecção da(s) chave(s) criptográficas (e.g. pode até considerar chaves fixas no próprio código). A ideia é que iremos num exercício abaixo tratar precisamente esse aspecto.
- obs: nas implementações fornecidas das classes `Client` e `ServerWorker`, não deverá ser necessário "mexer" muito para além do método `process`.

Protocolo de acordo de chaves DH

Relembre o protocolo de acordo de chaves *Diffie_Hellman*:

1. Alice → Bob : gx
2. Bob → Alice : gy
3. Alice, Bob : Kmaster = g(x*y)

Onde g e p são os paâmetros públicos do protocolo; *(x, gx)* e *(y, gy)* são os pares de chaves de Alice e Bob respectivamente; e `Kmaster` é o segredo estabelecido pelo protocolo.

PROG: `Client_dh.py` e `Server_dh.py`

Pretende-se adaptar os programas realizados no ponto anterior para que a chave por eles utilizada resulte da execução do protocolo de acordo de chaves *Diffie-Hellman*. Para isso iremos fazer uso da funcionalidade oferecida pela biblioteca `cryptography`, utilizando a classe `dh`, que acabará por "esconder" a matemática apresentada atrás.

Algumas observações:

- Para gerar chaves *Diffie Hellman*, temos primeiro de criar um objeto DHParameters. Este objeto é que dispõe do método que permite gerar a chave privada, da qual se poderá obter a chave pública respectiva.
- Pode gerar os parâmetros recorrendo ao método `dh.generate_parameters` – note que este processo pode demorar algum tempo.
  - fixar os parâmetros *(p,g)*, em que p é o primo que define o corpo e g o gerador adoptado. Pode usar valores fixos para estes parâmetros, e.g.:

```
p = 0xFFFFFFFFFFFFFFFFC90FDAA22168C234C4C6628B80DC1CD129024E088A67CC74020BBEA63B139B22514A08798E34C
g = 2
```

- Em criptofrafia assimétrica, as chaves tem uma estrutura complexa que não podem ser directamente comunicadas ou lidas/gravadas em ficheiro (não são meros *arrays* de *bytes* como no caso das chaves simétricas usadas até agora). Torna-se assim necessário transformar essas chaves (que são objetos de classes *Python*) de/para *arrays* de *bytes* – i.e. **serializar** os objectos. Podem consultar a documentação relevante (aqui e aqui). Podem escolher entre os *encodings* `DER` ou `PEM`, sendo que a única distinção prática é que `DER` é um formato binário e `PEM` usa caracteres imprimíveis semelhante ao *base64*.
- Note que os segredos criptográficos requeridos devem ser derivados a partir de `Kmaster` com recurso a uma KDF (e.g. `HKDF`, conforme sugestão apresentada na documentação).