Comunicações seguras

Estabelecimento de um canal seguro de comunicações

Membros:
João Marques - 89234
Tomás Costa - 89016

Índice

Introdução

Foram fornecidos aos alunos ficheiros relevantes a concretização do projeto, nomeadamente uma implementação base de um cliente e servidor que já realizavam comunicações entre si, e alguns ficheiros extra para setup do ambiente virtual.
O objetivo do trabalho era conseguirmos tornar esta comunicação semelhante a ruido para intrusos que estivessem a ouvir as mensagens na rede, cifrando esta mesma.
O objetivo principal deste trabalho era demonstrar os mecanismos necessários para estabelecer um canal de comunicação seguro e portanto não levamos o performance como maior objetivo, pelo que a transferência de alguns ficheiros com tamanhos consideráveis podem ser mais demorados.

Nota: Criamos um ambiente virtual python usando o pipenv, pelo que a pasta do ambiente não se encontra no diretório raiz. No entanto, o mesmo pode ser criado através dos comandos pipenv install, e executado usando pipenv shell. Os ficheiros adicionais (Pipfile e Pipfile.lock) pertencem ao pipenv.

Planeamento da segurança das comunicações

A arquitetura inicial da solução consistia na troca de mensagens entre um cliente e um servidor. No entanto, as mensagens circulavam de maneira aberta pela rede, vulneráveis a serem captadas por um elemento externo.

Dessa maneira, o nosso objetivo era implementar um mecanismo de encriptação baseado em chaves simétricas, partilhadas pelos 2 indivíduos envolvidos no processo.

A chave teria que ser conhecida pelos 2 processos, sendo privada relativamente a qualquer elemento externo, pelo que usámos o algoritmo de Diffie-Hellman para a troca privada de chaves.

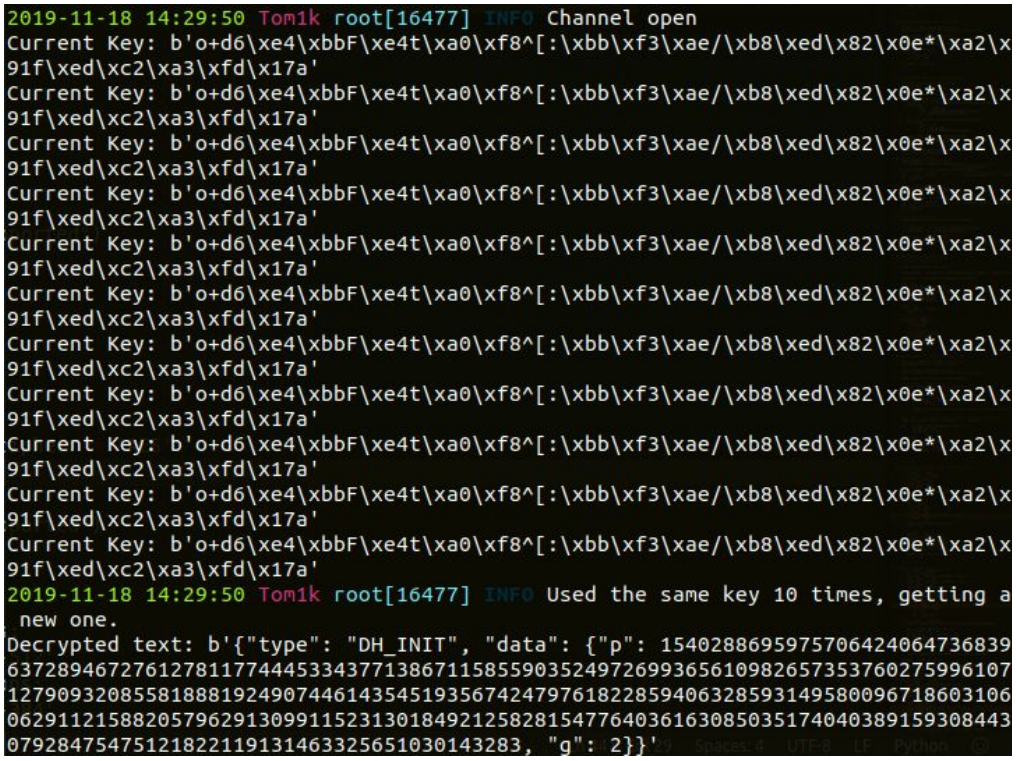
Quando a chave já é conhecida, todas as mensagens são cifradas, todas as mensagens são cifradas, sendo envolvidas noutra mensagem para o destinatário, juntamente com outros campos que possam ser necessários para a decifra (como o IV):

{
    'type' : 'SECURE_MSG'
    'data' : <encrypted message>
    'iv' : <iv>
}

Chunk desencriptado (data):

Decidimos numa fase final que iriamos também mudar a chave que estão a usar para cifrar/decifrar pois em ficheiros de grande dimensão, são precisas múltiplas cifragens e quanto maior o uso de uma chave, menor a sua segurança. Logo, ao fim de N iterações mudamos a chave e comunicam outra vez a negociação de chaves. Para não perdermos informação guardamos num ficheiro o que ainda não foi lido, e assim que estivermos a comunicar com a nova chave, resumimos a leitura desse ficheiro e, encriptamos com a nova chave. Na variável KEY_TTL guardamos o N de iterações antes de alterar a chave, num caso mais apropriado fariamos uma análise performance/segurança para chegarmos ao melhor número de iterações antes de trocarmos de chave. Mas no âmbito deste trabalho é suficiente mostrarmos o mecanismo, e portanto, o KEY_TTL assenta nas 70 iterações (mau usarmos a mesma chave 70 vezes na prática)

Rotação de chaves com N=10 (Client):



```
2019-11-18 14:29:50 Tom1k root[16477] INFO Channel open
Current Key: b'o+d6\xe4\xbbF\xe4t\xa0\xf8^[:\xbb\xf3\xae/\xb8\xed\x82\x0e*\xa2\x
91f\xed\xc2\xa3\xfd\x17a'
Current Key: b'o+d6\xe4\xbbF\xe4t\xa0\xf8^[:\xbb\xf3\xae/\xb8\xed\x82\x0e*\xa2\x
91f\xed\xc2\xa3\xfd\x17a'
Current Key: b'o+d6\xe4\xbbF\xe4t\xa0\xf8^[:\xbb\xf3\xae/\xb8\xed\x82\x0e*\xa2\x
91f\xed\xc2\xa3\xfd\x17a'
Current Key: b'o+d6\xe4\xbbF\xe4t\xa0\xf8^[:\xbb\xf3\xae/\xb8\xed\x82\x0e*\xa2\x
91f\xed\xc2\xa3\xfd\x17a'
Current Key: b'o+d6\xe4\xbbF\xe4t\xa0\xf8^[:\xbb\xf3\xae/\xb8\xed\x82\x0e*\xa2\x
91f\xed\xc2\xa3\xfd\x17a'
Current Key: b'o+d6\xe4\xbbF\xe4t\xa0\xf8^[:\xbb\xf3\xae/\xb8\xed\x82\x0e*\xa2\x
91f\xed\xc2\xa3\xfd\x17a'
Current Key: b'o+d6\xe4\xbbF\xe4t\xa0\xf8^[:\xbb\xf3\xae/\xb8\xed\x82\x0e*\xa2\x
91f\xed\xc2\xa3\xfd\x17a'
Current Key: b'o+d6\xe4\xbbF\xe4t\xa0\xf8^[:\xbb\xf3\xae/\xb8\xed\x82\x0e*\xa2\x
91f\xed\xc2\xa3\xfd\x17a'
Current Key: b'o+d6\xe4\xbbF\xe4t\xa0\xf8^[:\xbb\xf3\xae/\xb8\xed\x82\x0e*\xa2\x
91f\xed\xc2\xa3\xfd\x17a'
Current Key: b'o+d6\xe4\xbbF\xe4t\xa0\xf8^[:\xbb\xf3\xae/\xb8\xed\x82\x0e*\xa2\x
91f\xed\xc2\xa3\xfd\x17a'
Current Key: b'o+d6\xe4\xbbF\xe4t\xa0\xf8^[:\xbb\xf3\xae/\xb8\xed\x82\x0e*\xa2\x
91f\xed\xc2\xa3\xfd\x17a'
2019-11-18 14:29:50 Tom1k root[16477] INFO Used the same key 10 times, getting a
 new one.
Decrypted text: b'{"type": "DH_INIT", "data": {"p": 1540288695975706424064736839
63728946727612781177444533437713867115855903524972699365610982657353760275996107
12790932085581888192490744614354519356742479761822859406328593149580096718603106
06291121588205796291309911523130184921258281547764036163085035174040389159308443
07928475475121822119131463325651030143283, "g": 2}}'
```

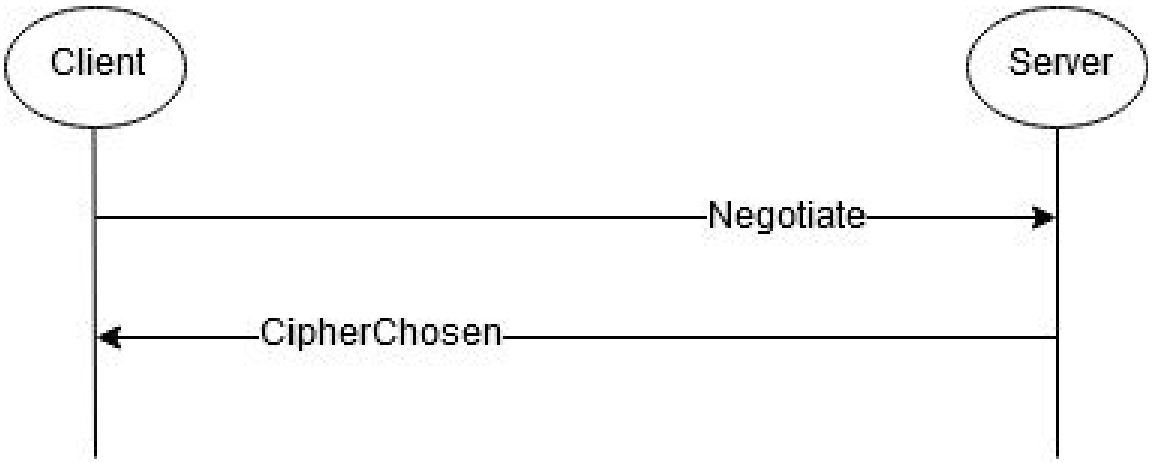
Negociação de chaves

Para a negociação de chaves decidimos que a melhor solução era o cliente anunciar quais as cifras, métodos e função de hashing é que suportava e enviar esta informação para o servidor. Assim que o servidor recebe essa mensagem, escolhe o melhor baseando-se em alguns critérios: melhor desempenho (visto que o processamento e server side), tentar a melhor segurança.

Fizemos esta escolha pois o servidor é mais confiável e pode adaptar melhor e portanto obter o melhor desempenho das disponíveis. Assim que for feita uma escolha, o servidor envia qual as especificações da chave a usar no futuro e o cliente guarda essa informação.

Na situação da troca de chaves, a informação da chave é atualizada e ambos o cliente e servidor começam a encriptar e desencriptar com a nova chave, respetivamente.

Diagrama Flow:



Cliente anuncia modos, servidor escolhe o melhor:

```
Connection from ('127.0.0.1', 43262)
2019-11-18 14:38:24 Tom1k root[17730] INFO Cipher chosen from message: {'type':
'NEGOTIATE', 'ciphers': ['AES', '3DES', 'ChaCha20'], 'modes': ['CBC', 'GCM', 'EC
B'], 'sinteses': ['SHA-256', 'SHA-384', 'SHA-512']}
2019-11-18 14:38:35 Tom1k root[17730] INFO File open
Decrypted text: b'{"type": "DATA", "data": "YWlvLXRjcHNlcnZlcj49MC4wLjMKY3J5cHRv
Z3JhcGh5Pj0yLjUKY29sb3JlZGxvZ3M9PTEwLjAK"}'
Decrypted text: b'{"type": "CLOSE"}'
```

Confidencialidade

Para garantir a confidencialidade na troca de chaves, usamos o algoritmo de Diffie-Hellman.

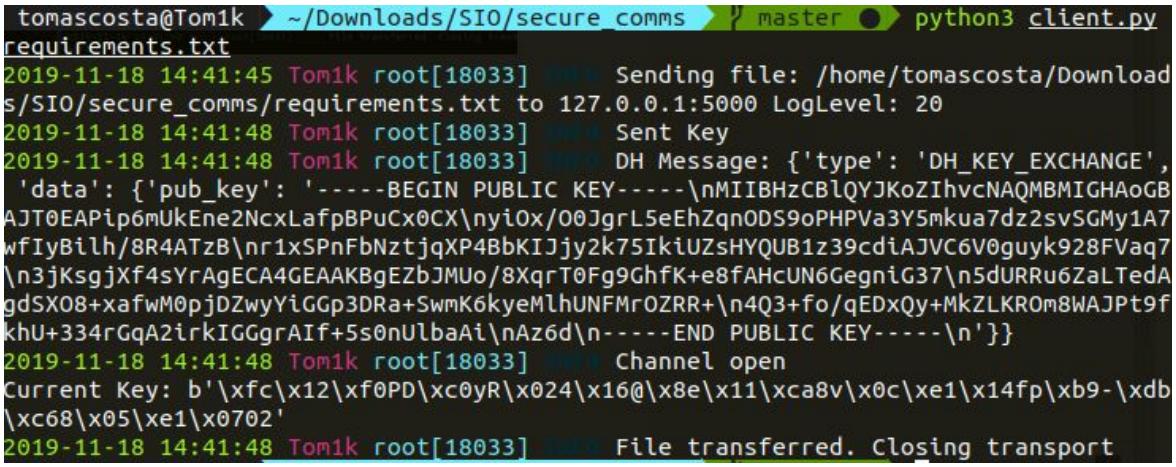
Este algoritmo baseia-se na troca de variáveis entre 2 sujeitos com o objetivo de chegar a uma chave comum aos 2 indivíduos, e que não possa, de maneira nenhuma, ser obtida através dos valores publicamente partilhados para chegar a essa chave.

Para esta implementação, usámos algumas informações da livraria Cryptography.io.

Começamos por gerar um valor privado em cada indivíduo, a que chamamos private key (não relacionado com private e public keys de cifras simétricas).
A partir deste valor, é gerado outro valor público (public key) que resulta da aplicação dos parâmetros públicos previamente partilhados entre os 2.
Finalmente, após o intercâmbio das public keys, geramos uma shared key que resulta dos valores da private key, junto com a public key.

Essa shared key é depois derivada, usando a função HKDF, com recurso ao algoritmo de síntese negociado anteriormente.

Partilha da chave pública:



Integridade

Para garantir integridade usamos um mecanismo de MIC que, resumidamente adiciona um campo extra ao final da mensagem com o hashing inicial. Do lado do cliente quando o recebemos, fazemos o hashing e comparamos com o campo que vem na mensagem, caso estes dois valores sejam diferentes, a integridade da mensagem foi violada e portanto lançamos uma exceção.

Conclusão

Com este trabalho, conseguimos criar um canal de comunicações seguro usando vários conceitos lecionados nas aulas teóricas e práticas. Com isto conseguimos impedir ataques de Man in the Middle (Eves), pois alguém que esteja a dar eavesdrop a conversa, não vai conseguir distinguir as mensagens que estão a ser passadas de ruído.
Estamos bastante satisfeitos com o trabalho final e pensamos cumprir os objetivos definidos no guião.