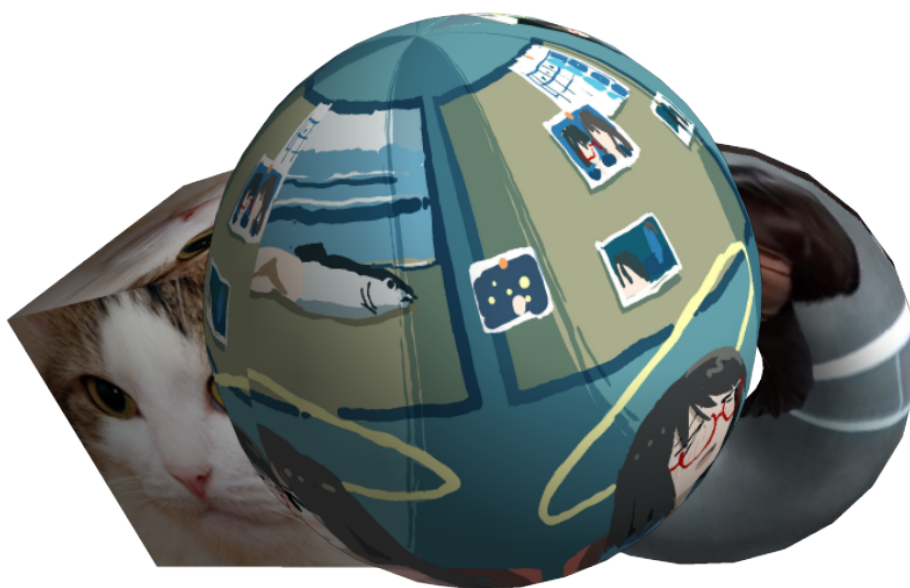




ESCOLA PROFISSIONAL DE LEIRIA

PROVA DE APTIDÃO PROFISSIONAL

DesIM - Mensagens Instantâneas numa Rede
Descentralizada



Guilherme Damásio Henriques

Nº - 216037

Gestão de Equipamentos Informáticos

João Cardoso

Resumo

O DesIM é uma rede de mensagens instantâneas que funciona num modelo par-a-par sobre a Internet. A Rede é composta por dois programas escritos em python. Um cliente (client), que será o programa usado pelo utilizador para comunicar na rede. E um Nó (node) que irá comunicar com outros Nós na rede. O cliente tem de se conectar a um Nó para comunicar na rede. Os clientes da rede usam autenticação por chave publica, efetuando a troca de chaves quando se ligam ao Nó e quando enviam uma mensagem direta por entre Nós para um cliente. O cliente ainda têm a opção de enviar mensagens a todos os clientes no mesmo Nó.

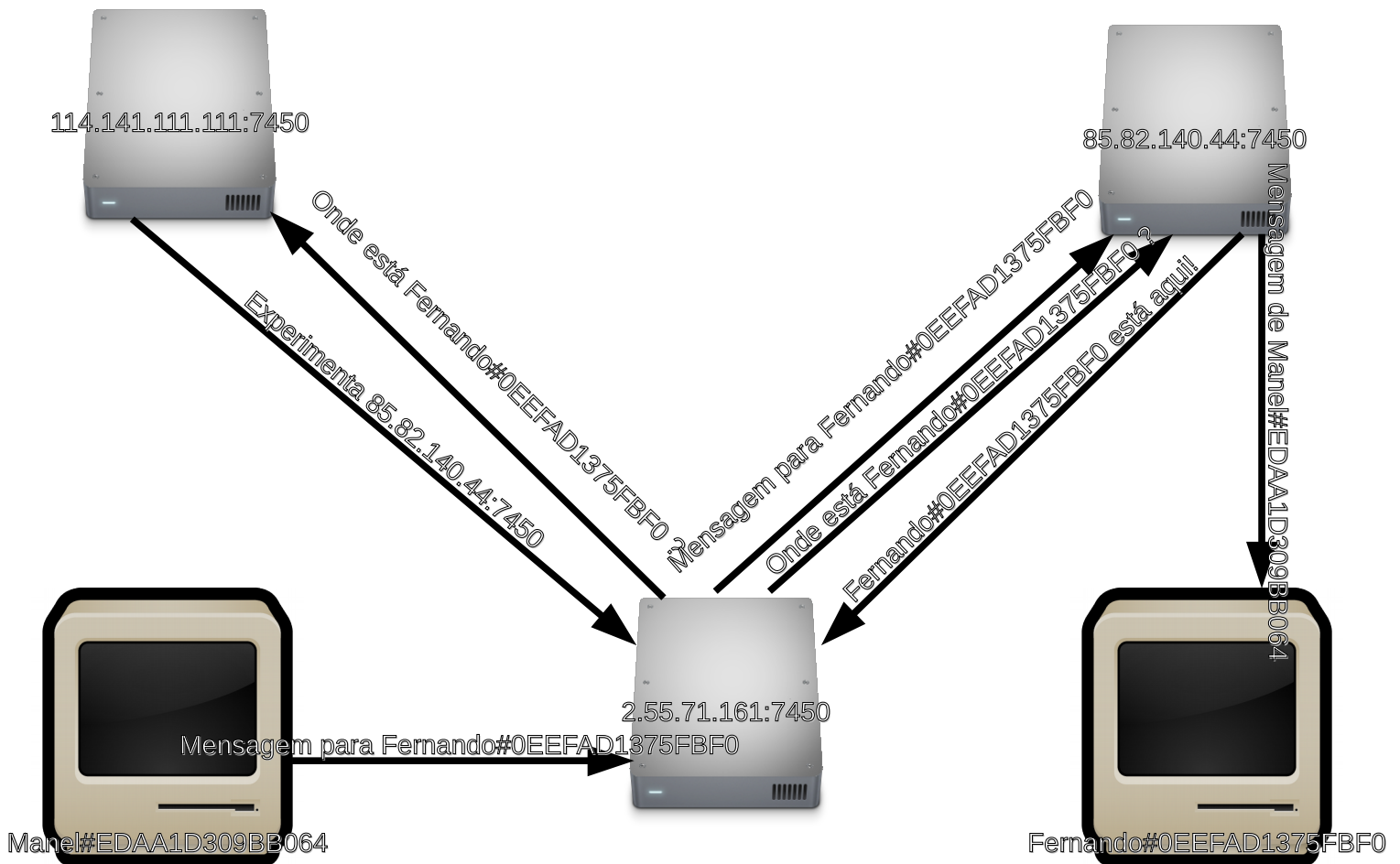


Figura 1 – Exemplo de do protocolo (abreviado)

Introdução

Enquadramento

Atualmente, a maior parte da comunicação por mensagens instantâneas na Internet é feita através de redes ou serviços que seguem a arquitectura cliente-servidor. Estes serviços baseiam-se no conceito que toda a informação e funções da rede é disponibilizada pelo servidor. Cabe a este garantir que as ações tomadas por uma identidade na rede são genuínas. E assim por consequência a autoridade da rede é detida por apenas um sujeito. Com isto a comunicação entre 2 clientes está dependente da vontade de quem gere o serviço usado.

As redes *Peer-to-Peer* surgem opostas a este modelo. Estas conseguem garantir a comunicação entre dois pontos sem a necessidade de uma identidade central coordenadora. Comunicando os cliente directamente entre si.

Objectivos

Com a realização deste projecto tenciono desenvolver um par de programas para a criação de um serviço de mensagens instantâneas, que funcione sobre uma rede *TCP/IP*, e tenha uma arquitectura *Peer-to-Peer* ou *P2P*.

Metodologia

A realidade atual das redes locais é uma de *NAT* e *Firewalls*. A maior parte dos computadores que falam na Internet, não podem com a mesma complexidade ficar a espera de conexões que dela provêm. Isto, é principalmente feito por questões de segurança. Mas este facto estraga um pouco os planos as redes P2P. Que têm que usar táticas como *Relaying* e *Hole Punching* para contornar este obstáculo.

A rede que vou desenvolver utilizará *Relaying*. Tendo isto em consideração, esta tarefa será desempenhada por os Nós ou *nodes* (em

inglês). Por isso na rede DesIM deverá existir dois tipos de operadores, os clientes e os nós (ou *nodes*). O uso de *Relaying* tem como benefício de não revelar o endereço de um emissor de uma mensagem.

Para a realização do projecto usei a linguagem de programação Python3. Ela é de alto nível, orientada a objectos e tem tipos dinâmicos. Uma das características que a melhor define, é a sintaxe definida pela formatação, enquanto a maior parte das linguagens usam chavetas ou parênteses para delimitar uma classe ou função, a linguagem python usa a sua indentação para este objectivo.

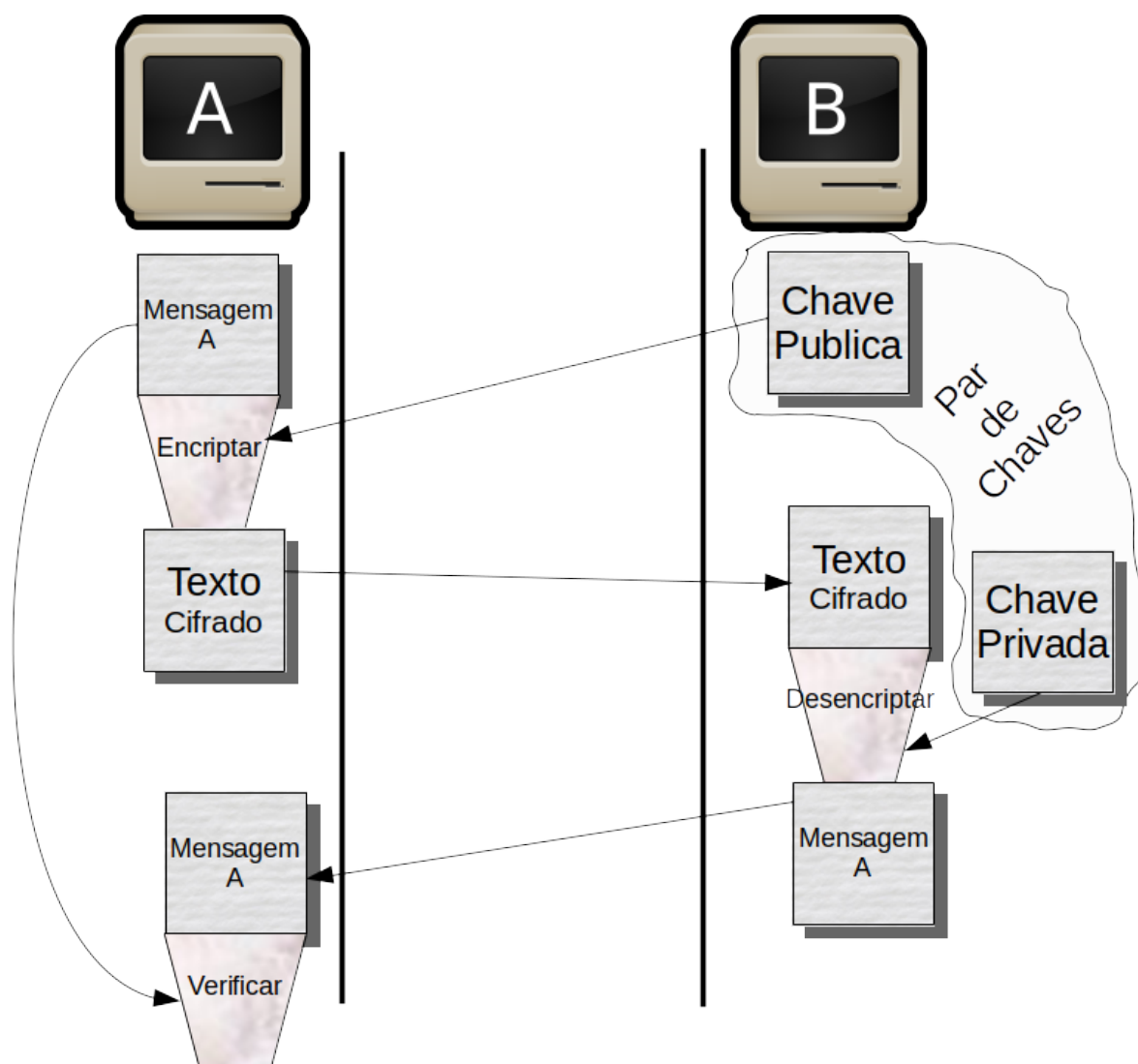


Figura 2 – Implementação de autenticação por par de chaves

A rede DesIM garante a descentralização da autenticação com o uso do modelo da encriptação assimétrica. Cada cliente irá ter um par de chaves, uma publica e outra privada. A chave publica só pode ser usada para encriptar conteúdo, e a privada para o desencriptar. Considerando isto, caso um cliente A queira verificar a autenticidade da identidade de outro B, este A irá requisitar que lhe seja enviado a chave publica de B. Com esta chave, A irá encriptar uma mensagem aleatória enviar o texto cifrado para B, o texto será desencriptado em B com a sua chave privada, B irá assim obter a mensagem criada por A. Agora B pode garantir a A que possui um par de chaves válido, sem sequer ter que revelar a sua chave privada.

Os Nós quando iniciados necessitam de coleccionar a localização de outros Nós disponíveis, para este efeito deve de ser usado um nó aleatório da rede. Isto é importante para garantir a descentralização da rede. O nó usado no primeiro contacto tem de ser definido manualmente.

O para facilitar a legibilidade da identidade dos clientes e minimizar os dados transmitidos na procura é usada nestes casos uma parte da hash da chave publica concatenada com a o nome do utilizador.

Os nós são indiferente aos tipos de dados que trocam nos pacotes de relay. Sendo estes pacotes usados para autenticação entre clientes e troca de mensagens, podendo as mensagens ser qualquer tipo de dados.

Fundamentos Teóricos

A comunicação entre pontos na rede DesIM é feita através de um socket TCP, na porta 7450. Selecionei esta porta por ser maior que 1024, e assim não exigir permissões de super-utilizador ou *root*, e também por não ser usada por outro protocolo comum na Internet. Um socket TCP garante a integridade dos dados e assim elimina a inconveniência de lidar com datagramas fragmentados ou desordenados em algoritmo. O protocolo TCP também oferece uma conveniência no facto que tem conexões duradouras através de uma maquina de estados. Isto permite facilitar a comunicação no segmento *Client-Node* devido a longevidade das conexões. Ou seja um cliente só irá se desconectar do Nó quando quiser sair da rede. Na comunicação entre Nós (*Nodes*) o protocolo UDP seria

mais adequado, exactamente por neste segundo segmento as conexões serem momentâneas a necessidade de processar um pedido.

Independentemente disto eu preferi usar um socket TCP para reduzir a complexidade do código e tornar o programa mais consistente.

Os dados enviados entre pontos devem ser encapsulados em JSON (JavaScript Object Notation). Escolhi este formato por ser popular e suportado em varias linguagens. Um campo obrigatório no objecto que contem um numero inteiro indica a operação a realizar. Cada operação tem campos associados a si mesma.

```
payload = {  
    op: 11,  
    tripcode: "Manel#EDAA1D309BB064",  
    forwarded: false,  
    node_searched: "",  
    time: 1562459511  
}
```

Figura 3 – exemplo de pacote encapsulado em JSON

Para além da linguagem base Python 3 tive de recorrer a alguns módulos ou livrarias do Python. As principais foram:

- os – Disponibiliza métodos para interagir com o sistema operativo.
- socket – É a implementação que sockets
- random – Interage com o gerador de números aleatórios do SO
- time - Acede a funções relacionadas com o tempo
- hashlib – biblioteca que implementa funções de hashing
- pycrypto – biblioteca que implementa funções criptográficas

Desenvolvimento

Implementação do protocolo Cliente - Nó

O tema deste projeto de aptidão profissional surgiu da reforma da seu conceito original. O plano inicialmente proposto tinha sido para a realização de um sistema para o armazenamento em rede. Ao longo do desenvolvimento pratico foi encontrado um problema fundamental com o a arquitectura pretendida. Com isto decidi aproveitar parte da ideia, reestruturando parte código fonte adaptei-o as necessidades de uma rede descentralizada para a troca de mensagens instantâneas.

Partindo de um protótipo que responderia a ações representadas em JSON. Comecei por redefinir quais seriam as operações por elas representadas. Preocupei-me primeiro com o segmento entre o cliente e o Nó.

Operações que o cliente pode enviar para o Nó:

Operação	Codigo	Descrição	Campos	
KeepAlive Response	0	Verifica se o socket socket está operacional	op - numero de operação client_time - timestamp do envio	
Auth	1	Usada na autenticação - são enviados dois pacotes JSON com o mesmo código de operação por esta ser um processo sequencial	Envio da chave publica	Envio da solução
			op - numero de operação tag - nome de utilizador public_key - chave publica	op - numero de operação solution - solução ao enigma criptografico
Message	2	Usada para enviar uma mensagem para o nó	op - numero de operação message - a mensagem a enviar	

			chan - uma indicação da localização ou canal recipient_tripcode - destinatario da mensagem client_time - timestamp do envio
Logout	3	Usado para informar o servidor que se irá desconectar	op - numero de operação client_time - timestamp do envio

Tabela 1 - Comandos do Cliente para o Nó

Assim também defini as operações que o nó pode fazer ao cliente:

Operação	Codigo	Descrição	Campos	
KeepAlive	0	Verifica se o socket socket está operacional	op - numero de operação server_time - timestamp do envio	
Auth	2	Usado na autenticação - pode conter o enigma criptografico ou informação sobre o sucesso da autenticação	Envio do enigma	Envio da solução
			op - numero de operação challenge - dados aleatórios encriptados com a chave publica do cliente	op - numero de operação auth_success - valor booleano com o estado da autenticação auth_msg - contexto do resultado da autenticação
Message	3	Para o cliente receber uma mensagem local ao nó	op - numero de operação message - a mensagem a enviar	

			chan - uma indicação da localização ou canal sender_tripcode - remetente da mensagem server_time - timestamp do envio
Disconnect	4	Usado para fechar o socket	op - numero de operação server_time - timestamp do envio

Tabela 3 - Comandos do Nó para o Cliente

Esta listagem de operações é a final, durante o desenvolvimento do projecto foram feitas varias alterações a estas tabelas.

Uma das modificações principais foram a adição das operações de *keepAlive*. Em testes os sockets tem potencial para se subitamente irresponsivos. Para detectar e lidar com estas falhas rapidamente é necessário que o socket seja regularmente usado.

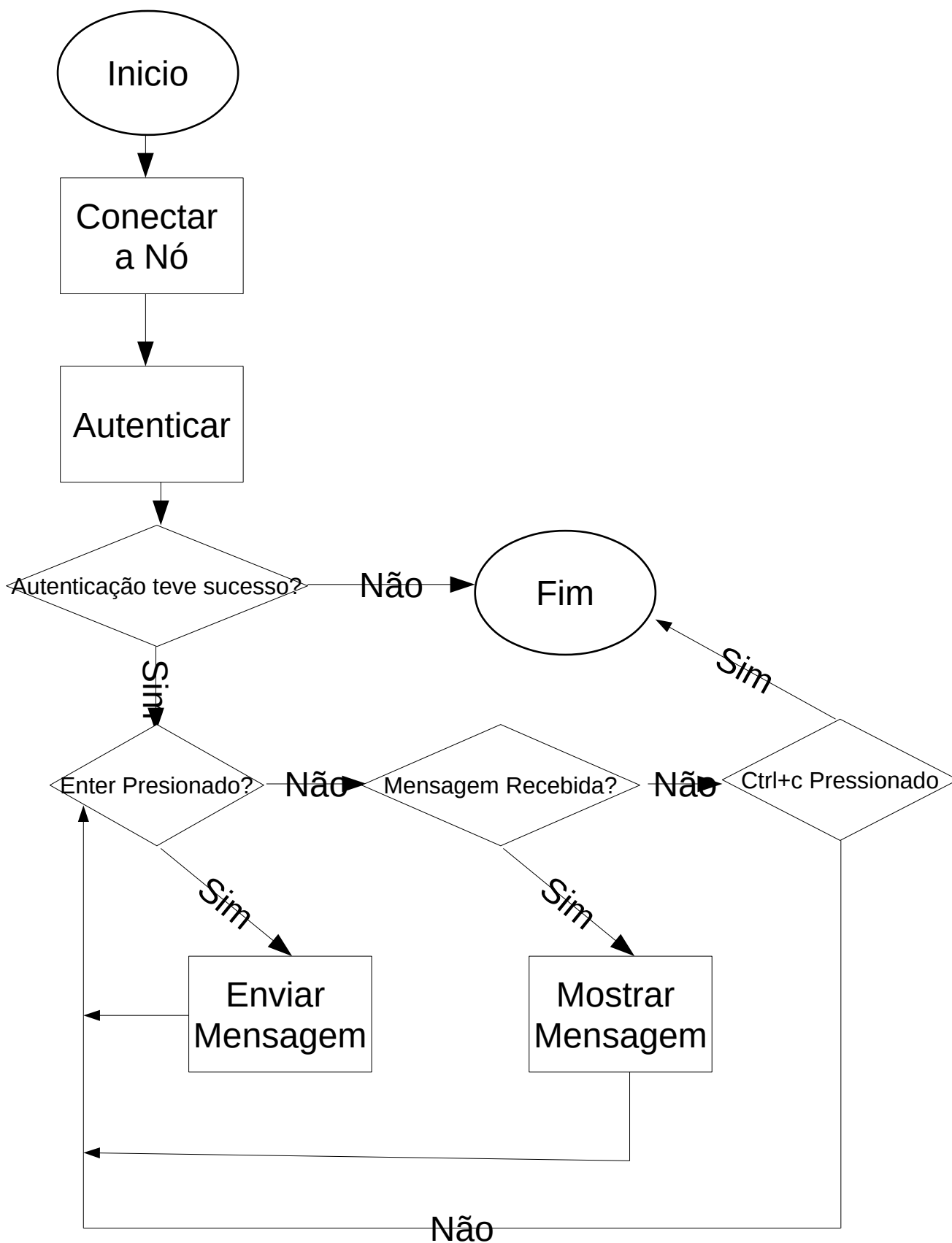
Também eliminei uma operação que o servidor poderia enviar que se chamava *peerStatusUpdate* esta operação seria usada para atualizar o cliente sobre o estado de outros clientes na rede.

Sobre Tripcodes

Tradicionalmente, os tripcodes são um metodo de autenticação descartavel usada em fóruns em que a identidade do autor de uma postagem só é relevante no mesmo fio ou tópico. Porém, a utilização de tripcodes neste projecto só é feita para diminuir a quantidade de dados transmitidos e tornar a identificação de utilizadores mais legível.

O Cliente

Com isto definido fui implementando estas operações no programa do cliente e no Nó sequencialmente. Ou seja na ordem em que estas seriam usadas. No fim fiquei com um algoritmo do cliente livremente representado pelo fluxograma seguinte.



Fluxograma 1 - Algoritmo geral do cliente.

O Nó

Na parte do Nó decidi que dividir certas funções em módulos seria crucial para a organização do código. Os módulos do Python funcionam como o “imports” ou #include noutras linguagens. Simplesmente incluem código de outros ficheiros no ficheiro que os importou. No Python o statement import irá primeiro procurar no directoria que o script se encontra a correr por outros ficheiros de com código fonte. O statement “import config” irá importar o código fonte de um ficheiro chamado config.py no mesmo directório. Então assim dividi o código fonte em vários ficheiros.

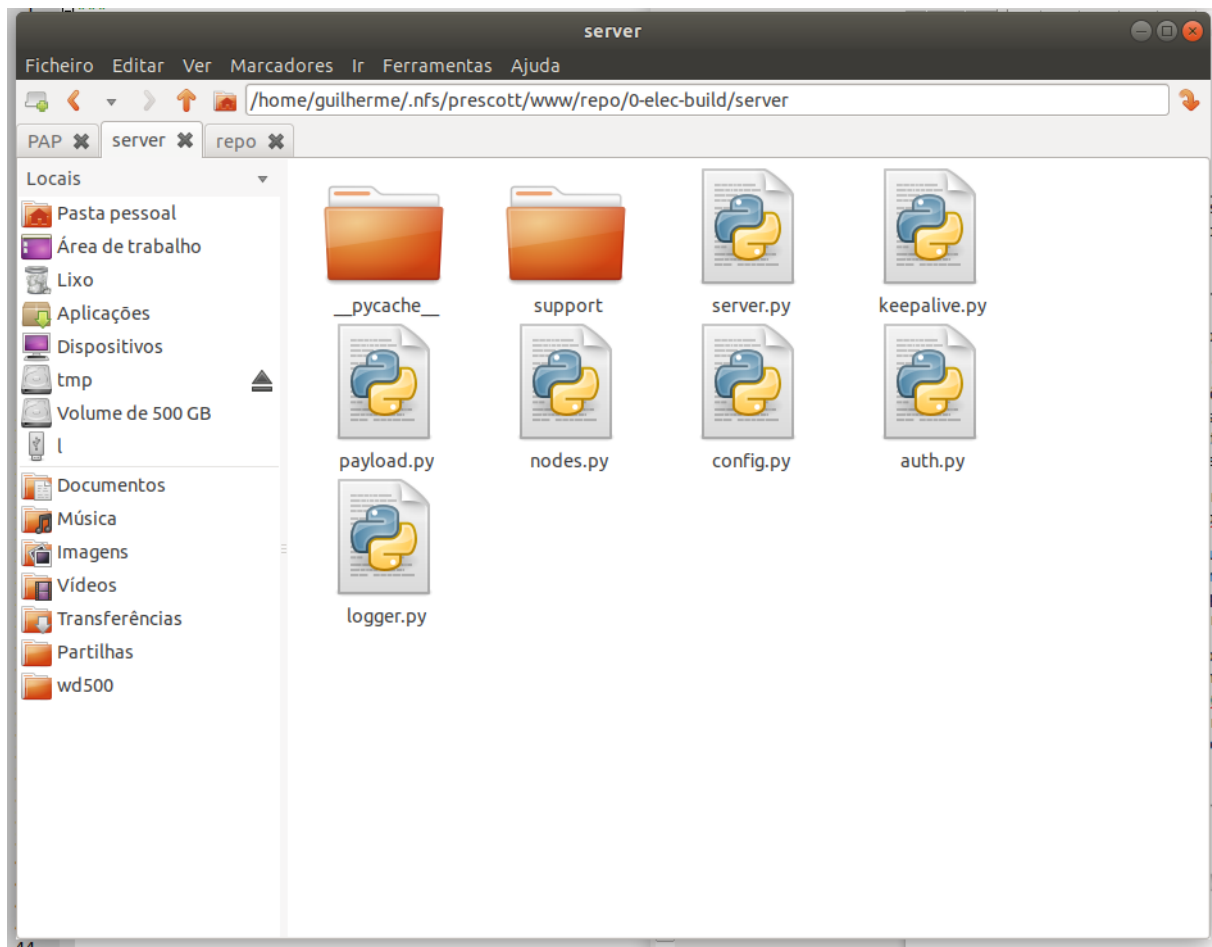


Figura 4 – Os modulos

- server.py – contem a função main() e é o que inicia e corre o node
- keepalive.py – contem o código que trata dos keepalives
- payload.py – constrói e decodifica os pacotes de JSON
- nodes.py – tem a lista de nodes conhecidos e funções para interagir com esta
- config.py – são definidas variáveis como a porta

- auth.py - trata de operações criptográficas e de autenticação
- logger.py - oferece uma forma programática de mudar o registo de eventos

```

while server_run:
    logger.p("accepting connections")

    (tmp_s, tmp_ip) = server.accept()

    wrapped = False
    search = True

    while search:
        logger.p("searching for slot")
        if not last_disconnected_slot == -1:
            server_full = False
            search = False
            logger.p("reusing", last_disconnected_client)
            conn_id = last_disconnected_slot
            last_disconnected_slot = -1
        else:
            if not server_full:
                if conn_id == config.MAX_SLOTS and wrapped == False:
                    wrapped = True
                    conn_id = 0
                elif conn_id == config.MAX_SLOTS and wrapped == True:
                    search = False
                    server_full = True
                    tmp_s.send(payload.client.mk_auth_response(-1).encode())
                    tmp_s.shutdown(socket.SHUT_RDWR)
                    tmp_s.close()
                    break
            else:
                if conn_arr[conn_id].active == False:
                    logger.p("found free slot", str(conn_id))
                    search = False
                else:
                    conn_id += 1

    if server_full == True:
        logger.p("Server has hit connection limit.")
        conn_id = 0
    else:
        conn_arr[conn_id] = peer(conn_id)
        (conn_arr[conn_id].socket, conn_arr[conn_id].ip) = (tmp_s, tmp_ip)

        conn_arr[conn_id].thread = threading.Thread(target = new_peer,
args=(conn_arr[conn_id],))
        conn_arr[conn_id].thread.start()
        conn_id += 1

shutdown_server()

```

Este algoritmo é o loop principal do nó ele fica a espera de conexões, e quando recebe uma ele aloca um espaço para guardar informação sobre o par. Depois de guardar um espaço é chamado um novo thread que irá ser encarregado da comunicação com a máquina remota. A função chamada, ou seja a `new_peer()`, não assume que o par conectado é um cliente enquanto este não usar a operação 1 para se autenticar. E enquanto este não estiver autenticado ele não pode fazer mais operações no Nó.

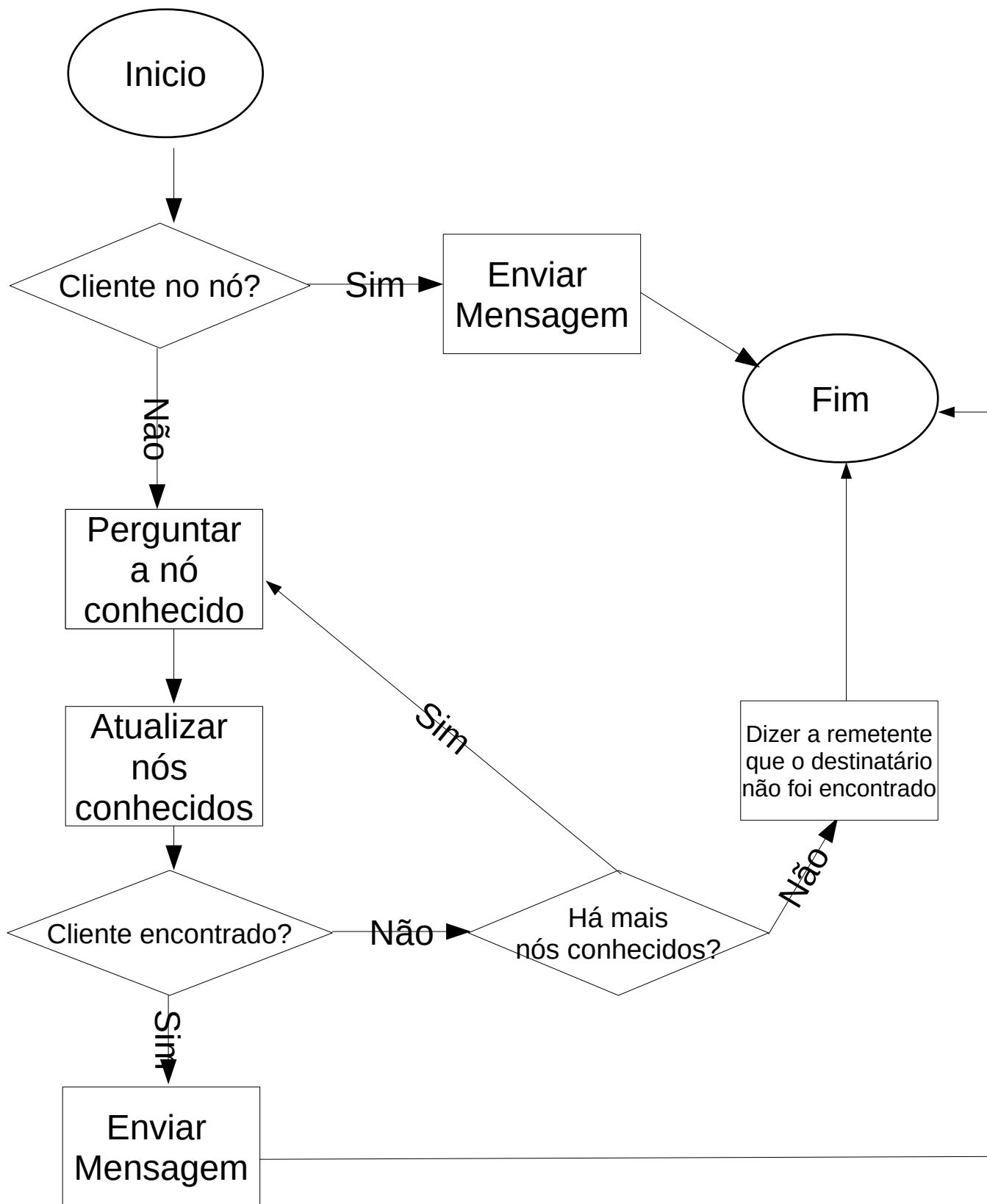
Comunicação entre *Nodes*

A parte que implementei de seguida foi a comunicação entre nó. Comecei novamente por definir quais seriam os pacotes a circular.

Operação	Código	Descrição	Campos
Lookup _request	11	Pergunta a um nó se ele tem um cliente.	op - numero de operação tripcode - identificação do cliente procurado time - timestamp do envio
Lookup _response	12	A resposta a pergunta anterior, caso a procura falhe é enviada a lista de nós conhecidos. Caso a procura tenha sucesso o nó não deve de fechar a conexão	op - numero da operação result - bool sobre a presença do cliente known_nodes - nodes conhecidos por este cliente time - timestamp do envio
Relay	13	Encaminha mensagens entre clientes	op - numero de operação sender_t - tripcode de remetente dest_t - tripcode de destinatário msg_obj - objecto da mensagem time - timestamp do envio

Tabela 3 – Comandos Entre Nós

Com isto decidido faltava apenas criar o algoritmo de procura de clientes.



Fluxograma 2 - Algoritmo geral da procura de clientes.

Este algoritmo irá primeiro procurar na lista de clientes, que estão conectados ao nó, por um destinatário. Caso este não exista, ele irá começar, pelo início da lista de outros nós conhecidos, a perguntar pela localização de um cliente. As respostas dadas por outros nós contém a lista de nós conhecidos deles, esta é usada para atualizar a sua lista de nós conhecidos, quando o nó chega ao fim da lista este pode assumir que o destinatário não existe.

Com isto implementado só faltava atualizar o código do cliente para este poder mandar mensagens diretas. Isto é feito com a prefixação da mensagem com `"/dm [utilizador] [mensagem]"`. Assim o cliente pode detectar o carácter `"/` como o início de um comando dar parse ou destinatário e mensagem.

Para além disto tive que adicionar uma segurança para remover caracteres de controlo como por exemplo o carriage return o null e outros para que o terminal de um destinatário não pudesse ser controlado remotamente.

```
guilherme@Haswell:~/nfs/prescott/www/repo/0-elec-build/server$ python3 auth.py
Python 3.6.7 (default, Oct 22 2018, 11:32:17)
[GCC 8.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import server
1562593679 ('created socket', <socket.socket fd=3, family=AddressFamily.AF_INET,
1562593679 ('bound',))
1562593679 ('listen',))
1562593679 ('accepting connections',))
1562593705 ('searching for slot',))
1562593705 ('found free slot', '0')
1562593705 ('accepting connections',))
1562593723 ('searching for slot',))
1562593723 ('found free slot', '1')
1562593723 ('accepting connections',))

guilherme@Haswell:~/nfs/prescott/www/repo/0-elec-build/server/support/testing$ python3 demo_client.py
Conectado ao destino.
{"op": 2, "auth_status": 0, "auth_msg": "Successfully logged in", "auth_success": true, "tripcode": "teste123#bc3e1edb7204f2"}
teste123@#Geral>ehlo
# ehlo
teste123 @ #Geral >teste123@#Geral>
```

Figura 5 – Rede em funcionamento

Conclusão

Objectivos Concretizados

O projecto DesIM foi implementado com sucesso. O algoritmo de procura de clientes entre nós garante a integridade da rede e permite que a troca de informação entre duas identidades seja feita descentralizadamente.

Limitações

Uma das partes em que este projecto podia ser melhorado era na adição de encriptação das mensagens baseada na chave publica do cliente.

Possibilidade de melhoramento

Certas partes do algoritmo podiam ser melhoradas para diminuir a redundancia de algumas operações.

Apreciação final

Este trabalho foi habilitador e ensinou-me sobre a importância do planeamento e estrutura de programas de computador.

Bibliografia

<https://docs.python.org/3/reference/>

<https://docs.python.org/3/library/socket.html>

<https://docs.python.org/3/library/threading.html>

https://en.wikipedia.org/wiki/Public-key_cryptography

<https://web.mit.edu/alexmv/6.037/sicp.pdf>

