Mestrado em Engenharia Informática (MEI)
Mestrado Integrado em Engenharia Informática (MiEI)

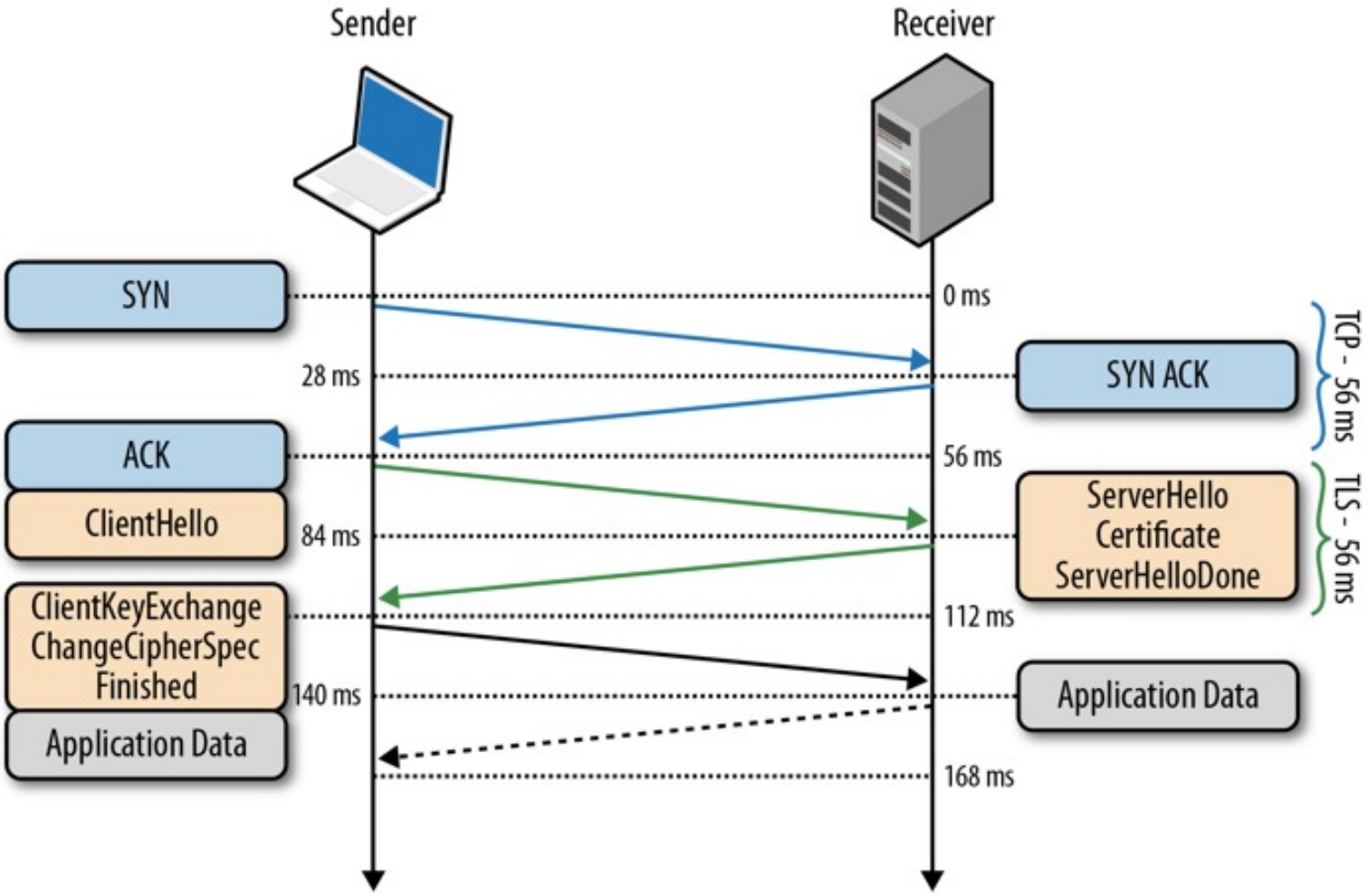
Perfil de Especialização **CSI** : Criptografia e Segurança da Informação

Engenharia de Segurança

Tópicos

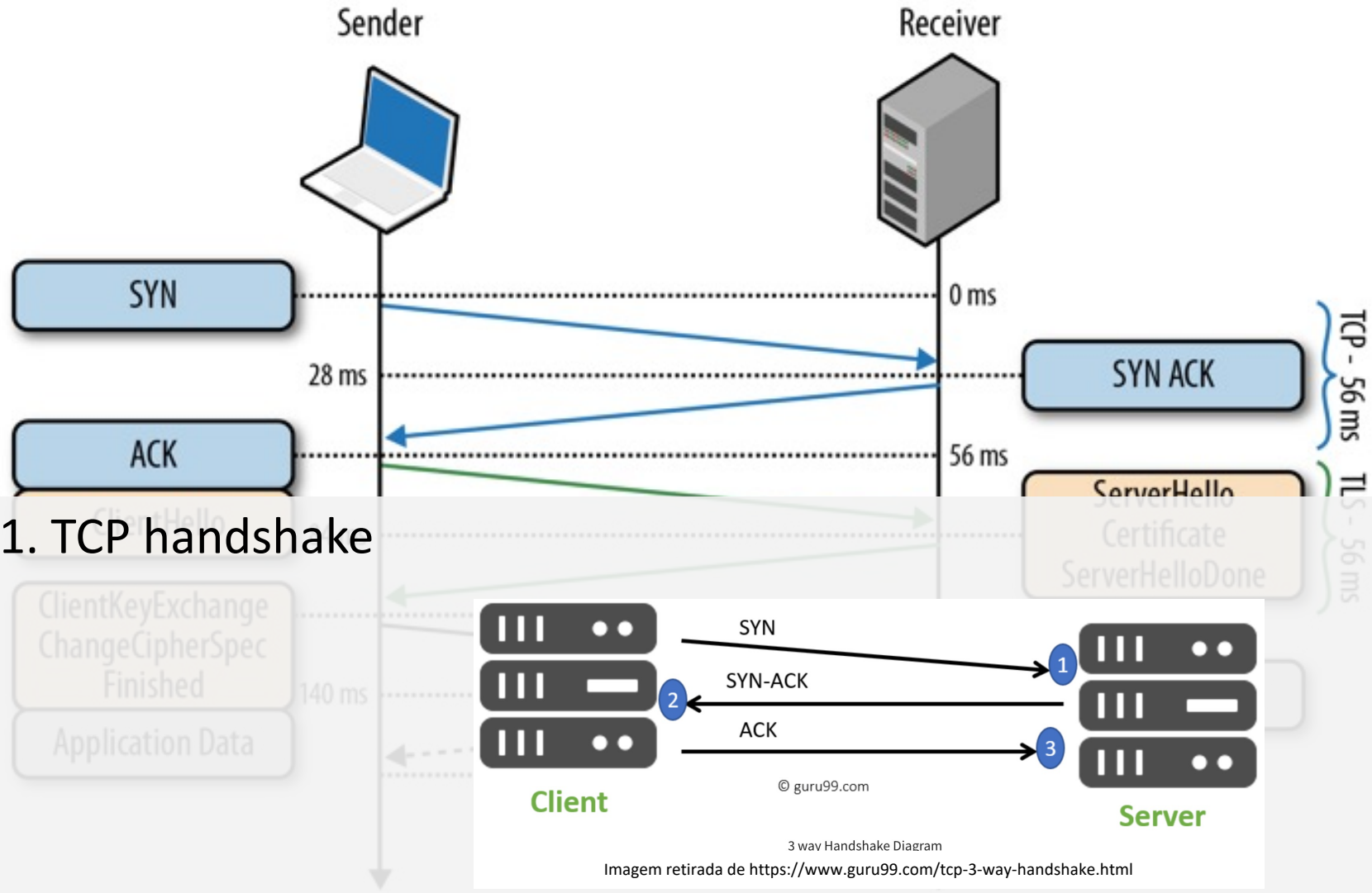
- **Parte X: Aplicações e protocolos**
  - **TLS**
  - SSH
  - TOR
  - *Blockchain*/DLT

SSL/TLS (RFC 5246)



- Nota: supondo um roundtrip de 56 ms

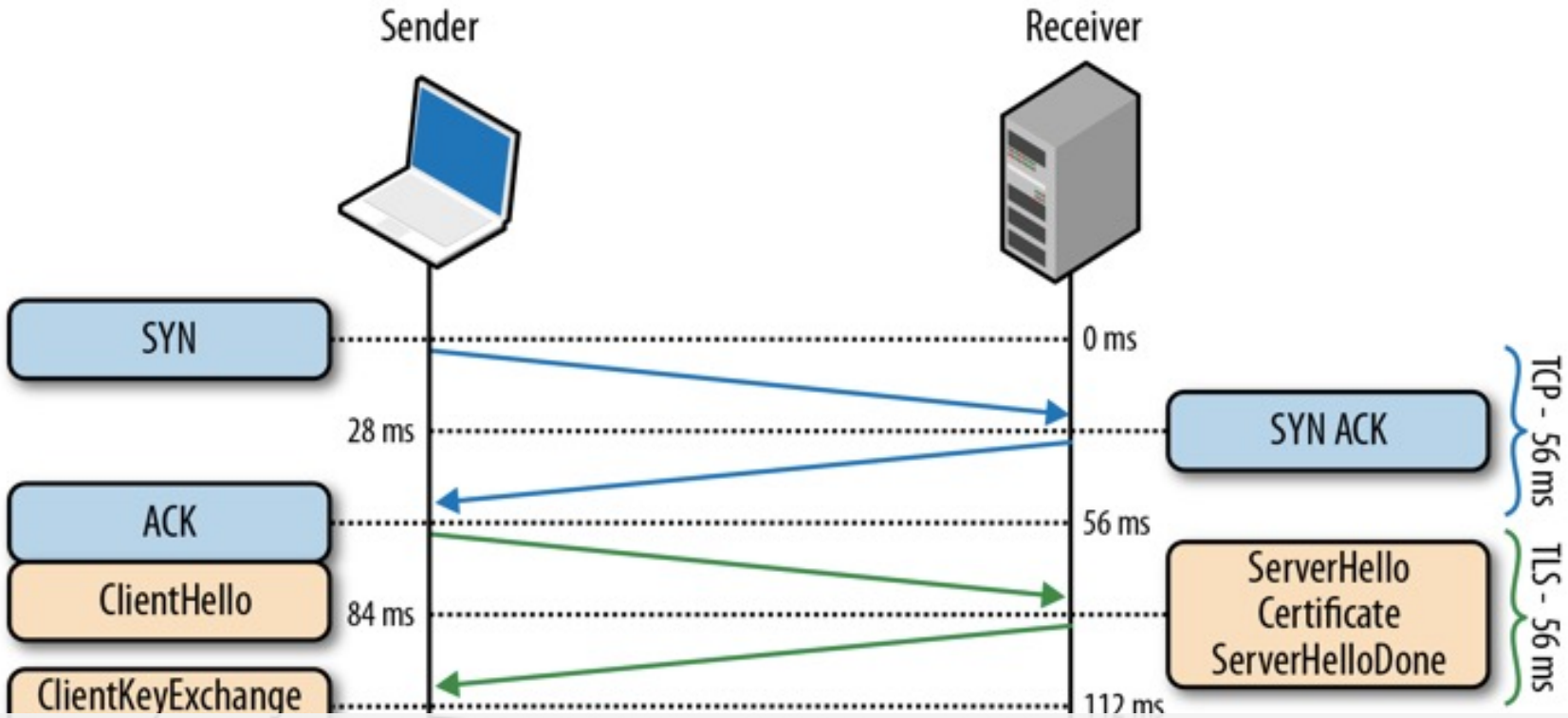
SSL/TLS (RFC 5246)



- 1. TCP handshake

Imagem retirada de https://www.guru99.com/tcp-3-way-handshake.html

SSL/TLS (RFC 5246)



- 2a. Cliente envia dados em plaintext, tais como versão do protocolo TLS, lista das cifras suportadas (por exemplo, TLS_RSA_WITH_AES_256_CBC_SHA256) e outras opções TLS;
- 2b. O servidor obtém a versão do protocolo TLS para comunicação, escolhe a cifra a utilizar da lista fornecida pelo cliente, adiciona o seu certificado e envia a resposta para o cliente. Opcionalmente pede o certificado do cliente.

jose.miranda@devisefutures.com

SSL/TLS (RFC 5246)

- 3a. Assumindo que ambas as partes negociaram uma versão comum do protocolo e cifras e, que o cliente "aceita" o certificado fornecido pelo servidor, o cliente inicia a troca de chaves (RSA ou Diffie-Hellman), utilizado para estabelecer a chave simétrica para a sessão;
- 3b. O servidor processa os parâmetros de troca de chaves enviada pelo cliente, valida a integridade da mensagem verificando o MAC e, devolve a mensagem "Finished" cifrada.

- 3c. O cliente decifra a mensagem com a chave simétrica negociada, verifica o MAC, e se tudo estiver correcto, estabelece o túnel (toda a comunicação passa a ser cifrada pela chave simétrica negociada) e os dados aplicacionais são enviados.

SSL/TLS – teste (www.ssllabs.com)

SSL Report: www.uminho.pt (193.137.9.114)

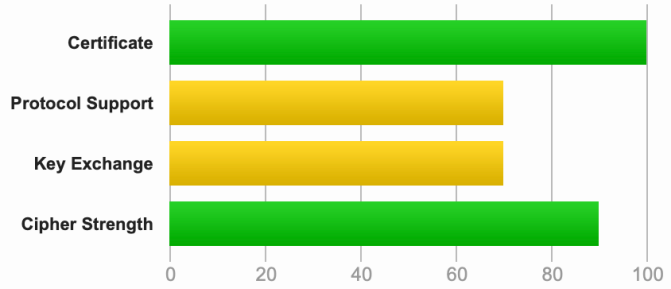
Assessed on: Thu, 14 Apr 2022 17:32:32 UTC | Hide | Clear cache

Scan Another »

Summary

Overall Rating

B



Visit our documentation page for more information, configuration guides, and books. Known issues are documented here.

This server supports weak Diffie-Hellman (DH) key exchange parameters. Grade capped to B. MORE INFO »

There is no support for secure renegotiation. MORE INFO »

This server supports TLS 1.0 and TLS 1.1. Grade capped to B. MORE INFO »

Certificate #1: RSA 4096 bits (SHA384withRSA)

Server Key and Certificate #1

Subject	*.uminho.pt Fingerprint SHA256: 578b42546de0527ba75c7f0f4ebde43203a7678dbda527c18faad8da7da42029 Pin SHA256: rYd3l++9lf1JPSocBFgUcBWPwqlJWoaMjS5PcncKqW4=

SSL/TLS – teste (www.ssllabs.com)

Configuration

Protocols

TLS 1.3	No
TLS 1.2	Yes
TLS 1.1	Yes
TLS 1.0	Yes
SSL 3	No
SSL 2	No

Cipher Suites

 # TLS 1.2 (suites in server-preferred order)

TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xc02f) ECDH secp384r1 (eq. 7680 bits RSA) FS	128
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (0xc013) ECDH secp384r1 (eq. 7680 bits RSA) FS **WEAK**	128
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 (0xc027) ECDH secp384r1 (eq. 7680 bits RSA) FS **WEAK**	128
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xc030) ECDH secp384r1 (eq. 7680 bits RSA) FS	256
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (0xc014) ECDH secp384r1 (eq. 7680 bits RSA) FS **WEAK**	256
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384 (0xc028) ECDH secp384r1 (eq. 7680 bits RSA) FS **WEAK**	256
TLS_RSA_WITH_AES_128_GCM_SHA256 (0x9c) **WEAK**	128
TLS_RSA_WITH_AES_128_CBC_SHA (0x2f) **WEAK**	128
TLS_RSA_WITH_AES_128_CBC_SHA256 (0x3c) **WEAK**	128
TLS_RSA_WITH_AES_256_GCM_SHA384 (0x9d) **WEAK**	256
TLS_RSA_WITH_AES_256_CBC_SHA (0x35) **WEAK**	256
TLS_RSA_WITH_AES_256_CBC_SHA256 (0x3d) **WEAK**	256
TLS_RSA_WITH_CAMELLIA_128_CBC_SHA (0x41) **WEAK**	128
TLS_RSA_WITH_CAMELLIA_256_CBC_SHA (0x84) **WEAK**	256
TLS_DHE_RSA_WITH_AES_128_GCM_SHA256 (0x9e) DH 2048 bits FS	128

SSL/TLS – teste (www.ssllabs.com)

Handshake Simulation

Android 2.3.7 No SNI [2]	RSA 4096 (SHA384)	TLS 1.0	TLS_RSA_WITH_AES_128_CBC_SHA	No FS
Android 4.0.4	RSA 4096 (SHA384)	TLS 1.0	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA	ECDH secp384r1 FS
Android 4.1.1	RSA 4096 (SHA384)	TLS 1.0	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA	ECDH secp384r1 FS
Android 4.2.2	RSA 4096 (SHA384)	TLS 1.0	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA	ECDH secp384r1 FS
Android 4.3	RSA 4096 (SHA384)	TLS 1.0	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA	ECDH secp384r1 FS
Android 4.4.2	RSA 4096 (SHA384)	TLS 1.2	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	ECDH secp384r1 FS
Android 5.0.0	RSA 4096 (SHA384)	TLS 1.2	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	ECDH secp384r1 FS
Android 6.0	RSA 4096 (SHA384)	TLS 1.2	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	ECDH secp384r1 FS
Android 7.0	RSA 4096 (SHA384)	TLS 1.2	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	ECDH secp384r1 FS
Android 8.0	RSA 4096 (SHA384)	TLS 1.2	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	ECDH secp384r1 FS
Android 8.1	RSA 4096 (SHA384)	TLS 1.2	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	ECDH secp384r1 FS
Android 9.0	RSA 4096 (SHA384)	TLS 1.2	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	ECDH secp384r1 FS
Baidu Jan 2015	RSA 4096 (SHA384)	TLS 1.0	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA	ECDH secp384r1 FS
BingPreview Jan 2015	RSA 4096 (SHA384)	TLS 1.2	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	ECDH secp384r1 FS
Chrome 49 / XP SP3	RSA 4096 (SHA384)	TLS 1.2	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	ECDH secp384r1 FS
Chrome 69 / Win 7 R	RSA 4096 (SHA384)	TLS 1.2	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	ECDH secp384r1 FS
Chrome 70 / Win 10	RSA 4096 (SHA384)	TLS 1.2	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	ECDH secp384r1 FS
Chrome 80 / Win 10 R	RSA 4096 (SHA384)	TLS 1.2	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	ECDH secp384r1 FS
Firefox 31.3.0 ESR / Win 7	RSA 4096 (SHA384)	TLS 1.2	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	ECDH secp384r1 FS
Firefox 47 / Win 7 R	RSA 4096 (SHA384)	TLS 1.2	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	ECDH secp384r1 FS
Firefox 49 / XP SP3	RSA 4096 (SHA384)	TLS 1.2	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	ECDH secp384r1 FS
Firefox 62 / Win 7 R	RSA 4096 (SHA384)	TLS 1.2	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	ECDH secp384r1 FS
Firefox 73 / Win 10 R	RSA 4096 (SHA384)	TLS 1.2	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	ECDH secp384r1 FS
Googlebot Feb 2018	RSA 4096 (SHA384)	TLS 1.2	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	ECDH secp384r1 FS
IE 7 / Vista	RSA 4096 (SHA384)	TLS 1.0	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA	ECDH secp384r1 FS

SSL/TLS – teste (openssl)

```
bash-3.2$ openssl s_client -state -servername www.uminho.pt -connect www.uminho.pt:443
CONNECTED(00000005)
SSL_connect:before/connect initialization
SSL_connect:SSLv3 write client hello A
SSL_connect:SSLv3 read server hello A
depth=2 C = US, ST = New Jersey, L = Jersey City, O = The USERTRUST Network, CN = USERTrust RSA Certification Authority
verify return:1
depth=1 C = NL, O = GEANT Vereniging, CN = GEANT OV RSA CA 4
verify return:1
depth=0 C = PT, ST = Braga, L = Braga, O = Universidade do Minho, CN = *.uminho.pt
verify return:1
SSL_connect:SSLv3 read server certificate A
SSL_connect:SSLv3 read server key exchange A
SSL_connect:SSLv3 read server done A
SSL_connect:SSLv3 write client key exchange A
SSL_connect:SSLv3 write change cipher spec A
SSL_connect:SSLv3 write finished A
SSL_connect:SSLv3 flush data
SSL_connect:SSLv3 read finished A
---
Certificate chain
 0 s:/C=PT/ST=Braga/L=Braga/O=Universidade do Minho/CN=*.uminho.pt
   i:/C=NL/O=GEANT Vereniging/CN=GEANT OV RSA CA 4
 1 s:/C=US/ST=New Jersey/L=Jersey City/O=The USERTRUST Network/CN=USERTrust RSA Certification Authority
   i:/C=US/ST=New Jersey/L=Jersey City/O=The USERTRUST Network/CN=USERTrust RSA Certification Authority
 2 s:/C=NL/O=GEANT Vereniging/CN=GEANT OV RSA CA 4
   i:/C=US/ST=New Jersey/L=Jersey City/O=The USERTRUST Network/CN=USERTrust RSA Certification Authority
---
```

SSL/TLS – teste (openssl)

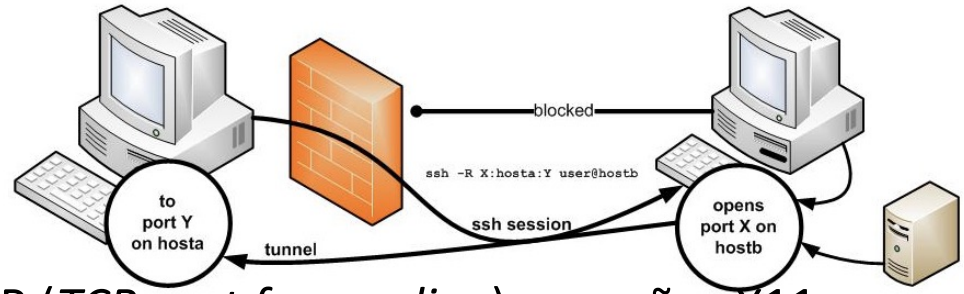
```
bash-3.2$ openssl s_client -state -servername www.uminho.pt -connect www.uminho.pt:443
CONNECTED(00000005)
SSL_connect:before/connect initialization
SSL_connect:SSLv3 write client hello A
SSL_connect:SSLv3 read server hello A
depth=2 C = US, ST = New Jersey, L = Jersey City, O = The USERTRUST Network, CN = USERTrust RSA Certification Authority
verify return:1
depth=1 C = NL, O = GEANT Vereniging, CN = GEANT OV RSA CA 4
verify return:1
depth=0 C = PT, ST = Braga, L = Braga, O = Universidade do Minho, CN = *.uminho.pt
verify return:1
SSL_connect:SSLv3 read server certificate A
SSL_connect:SSLv3 read server key exchange A
SSL_connect:SSLv3 read server done A
SSL_connect:SSLv3 write client key exchange A
SSL_connect:SSLv3 write change cipher spec A
SSL_connect:SSLv3 write finished A
SSL_connect:SSLv3 flush data
SSL_connect:SSLv3 read finished A
---
Certificate c
 0 s:/C=PT/ST
   i:/C=NL/O=
 1 s:/C=US/ST
   i:/C=US/ST
 2 s:/C=NL/O=
   i:/C=US/ST
---
---
No client certificate CA names sent
Server Temp Key: ECDH, P-384, 384 bits
---
SSL handshake has read 6173 bytes and written 376 bytes
---
New, TLSv1/SSLv3, Cipher is ECDHE-RSA-AES128-GCM-SHA256
Server public key is 4096 bit
Secure Renegotiation IS NOT supported
Compression: NONE
Expansion: NONE
No ALPN negotiated
SSL-Session:
    Protocol  : TLSv1.2
    Cipher    : ECDHE-RSA-AES128-GCM-SHA256
    Session-ID:
    Session-ID-ctx:
    Master-Key: 43D319672E3F54C8F69F4A46ABBD1FF51565852A74E5A98D091D8AF7A7CF01EBF1FB34C2181BD4549B617A148EE2DD12
    Start Time: 1649962962
    Timeout   : 7200 (sec)
    Verify return code: 0 (ok)
---
```

Tópicos

- **Parte X: Aplicações e protocolos**
  - TLS
  - **SSH**
  - TOR
  - *Blockchain*/DLT

SSH (Secure Shell)

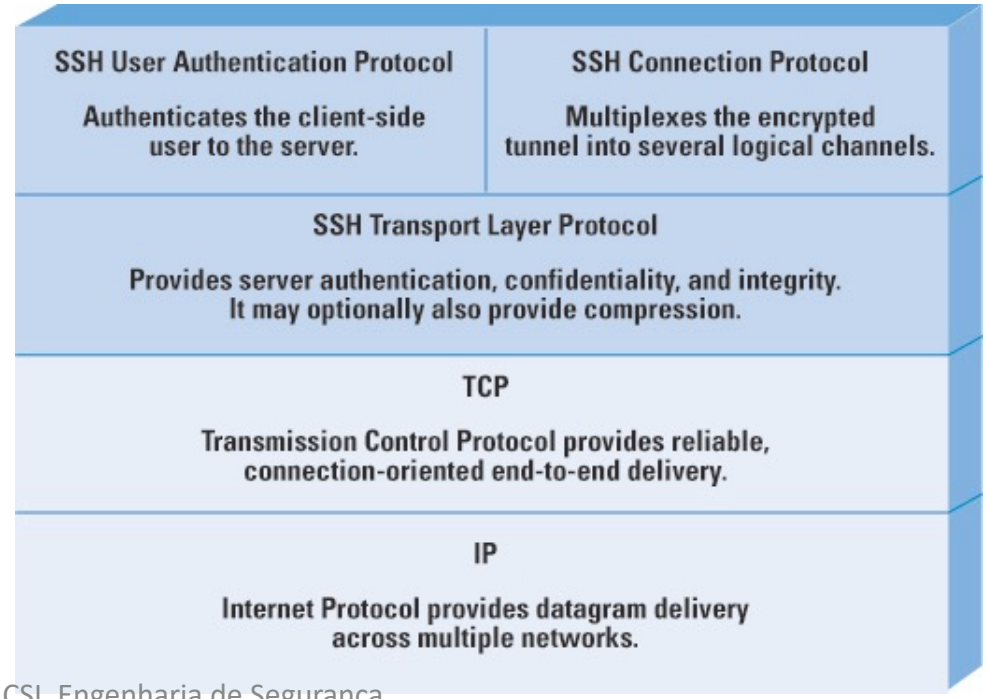
- **Protocolo criptográfico de rede** utilizado, entre outros, para:
  - Aceder a computador remoto,
  - Estabelecimento de túneis (tunneling) – por exemplo para criar um canal cifrado por onde transportar protocolos não cifrados (e.g. SMB),



  - Reencaminhamento de portas TCP (*TCP port forwarding*) e sessões X11,
  - Transferência de ficheiros através do SSH file transfer (SFTP) ou do secure copy (SCP),
  - Criação de VPN (para routeamento de pacotes entre redes diferentes ou efectuar bridge entre domínios de broadcast)
- Utiliza criptografia de chave pública para autenticar o computador remoto, assim como o utilizador, se necessário.

SSH (Secure Shell)

- Organizado em três protocolos sobre o TCP
  - SSH Transport Layer Protocol
    - Nota: tem a propriedade de *forward secrecy* (i.e., se uma chave for comprometida durante uma sessão, esse conhecimento não afecta a segurança das sessões anteriores)
  - SSH User Authentication Protocol
  - SSH Connection Protocol

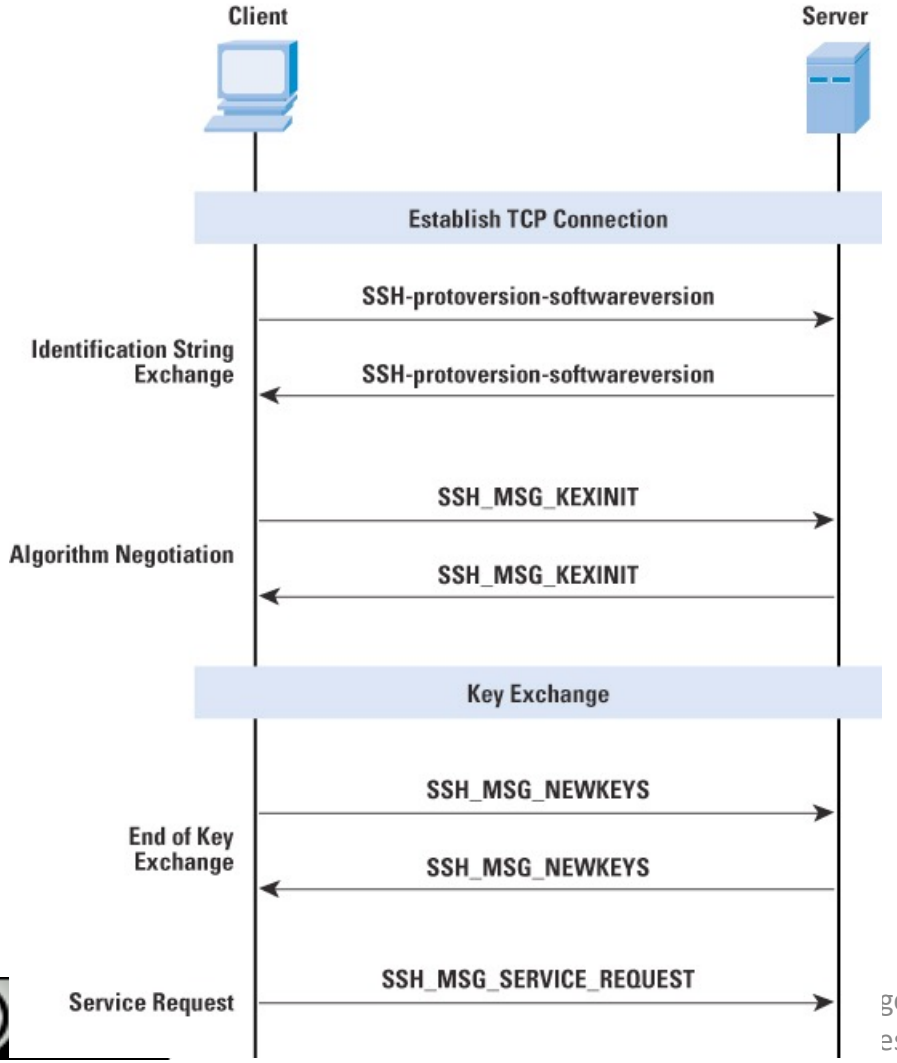


SSH (Secure Shell)

- SSH Transport Layer Protocol (RFC 4253)

SSH Transport Layer Protocol

Provides server authentication, confidentiality, and integrity. It may optionally also provide compression.



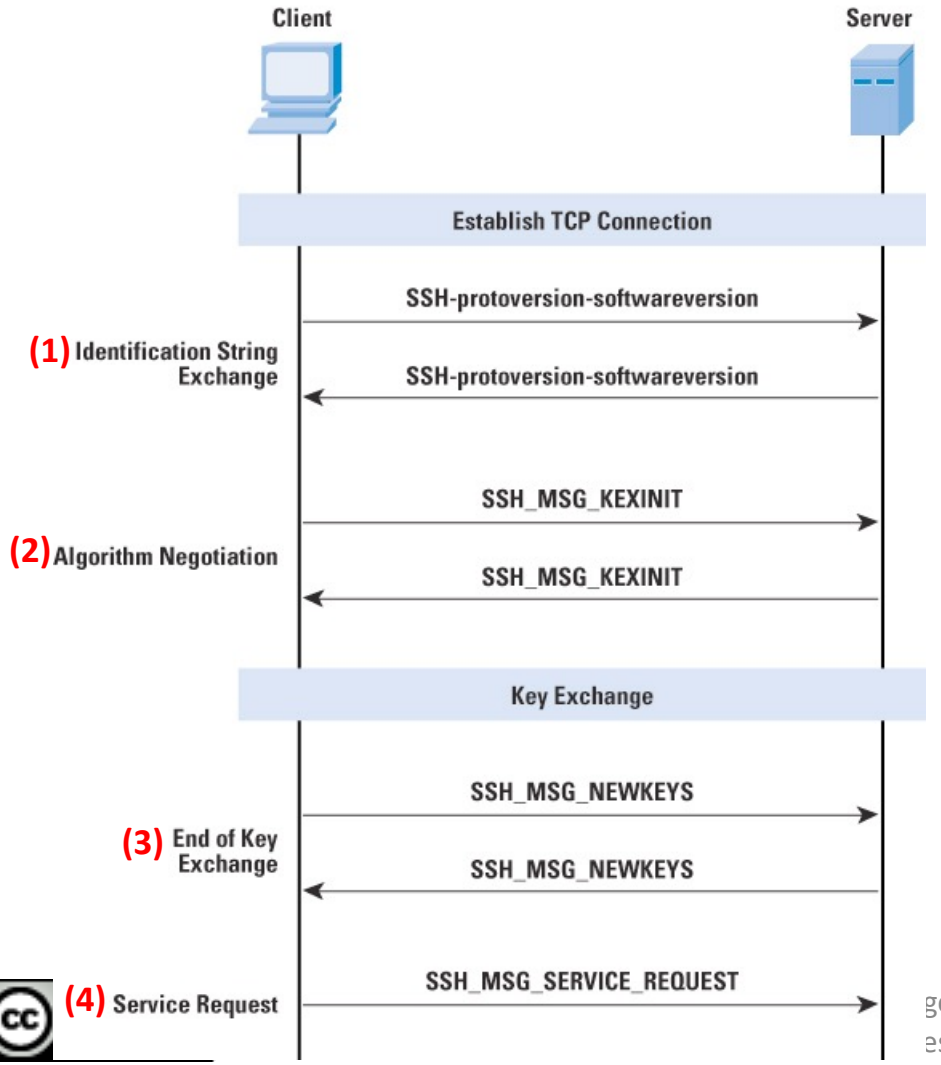
- A autenticação do servidor é efectuada baseada no seu par de chaves pública-privada;
- Para esta autenticação ser possível, o cliente tem que conhecer (previamente) a chave pública do servidor
  - O mais normal é o cliente ter uma base de dados local com as chaves públicas dos servidores conhecidos, não necessitando de uma infra-estrutura de administração central nem de coordenação com terceira parte (há a alternativa de a chave pública estar certificada por uma Entidade de Certificação e nesse caso o cliente tem que guardar o certificado de raiz da EC para validar os certificados com a chave pública dos servidores);

SSH (Secure Shell)

- SSH Transport Layer Protocol (RFC 4253)

SSH Transport Layer Protocol

Provides server authentication, confidentiality, and integrity. It may optionally also provide compression.



Passo 1: Troca de identificação, no formato *SSH-protoversion-softwareversion SP comments CR LF*

Passo 2: Negociação de algoritmos (troca de chaves - DH -, cifra - 3des-cbc, aes128-cbc, ... - , MAC - hmac-sha1, ... - e compressão - nenhuma ou zlib -), por ordem de preferência (escolhido o primeiro algoritmo indicado pelo cliente que o servidor também suporte, por cada tipo de algoritmo)

Passo 3: Troca de chave Diffie-Hellman, de acordo com RFC 2409, estabelecendo a chave de sessão e garantindo a autenticação do servidor (servidor utilizou a sua chave privada para assinar/cifrar a sua parte da troca de chaves DH). A partir deste momento, o tráfego passa a ser cifrado, conforme próximo slide.

Passo 4: o cliente envia um pacote SSH_MSG_SERVICE_REQUEST, solicitando o protocolo *SSH User Authentication* ou *SSH Connection*

SSH (Secure Shell)

- Pacotes SSH no segmento de dados do TCP

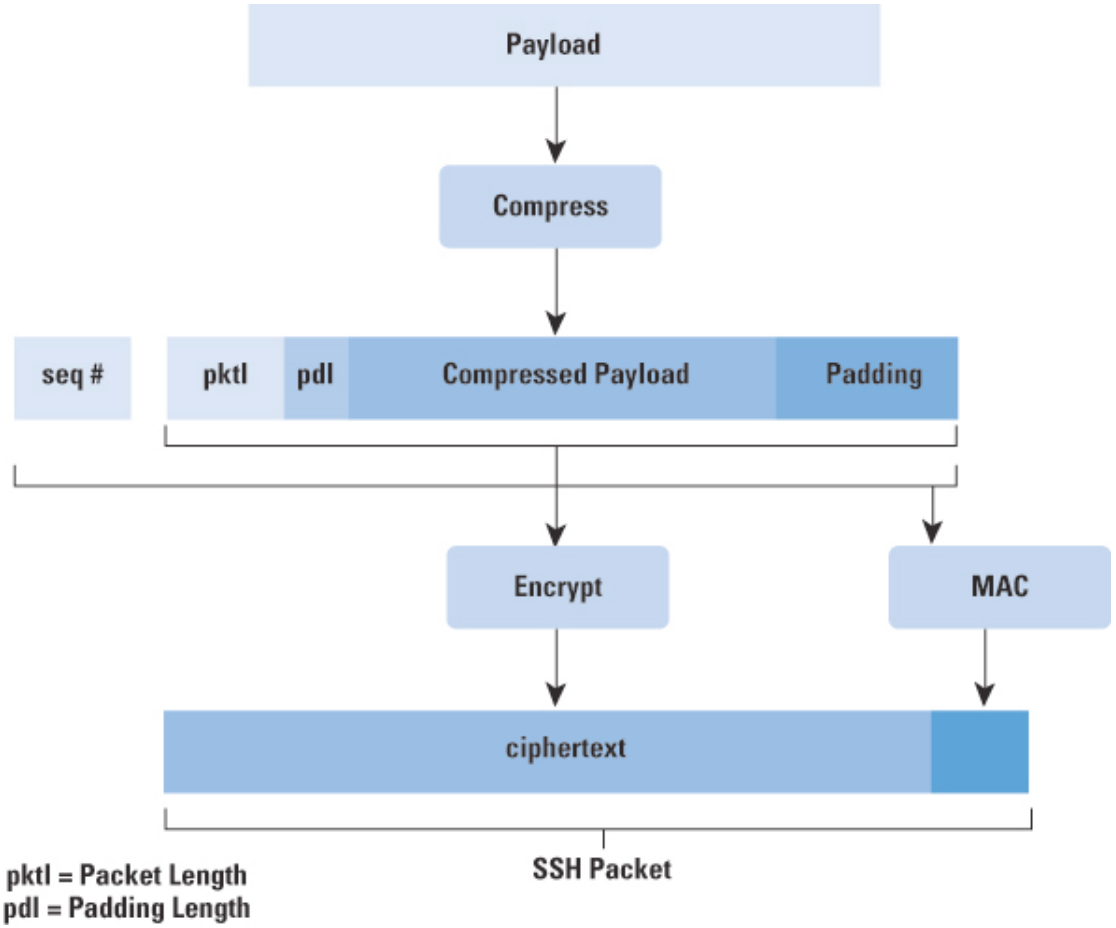
Packet length: tamanho do pacote SSH em bytes, não incluindo os campos *packet length* e MAC.

Padding length: tamanho do campo Padding.

Payload: constitui conteúdo útil do pacote SSH.

Padding: Contém bytes aleatórios de padding, de modo a que o tamanho total do pacote SSH (excluindo o MAC) é múltiplo de tamanho do bloco de cifra, ou 8 bytes se for uma *stream cipher*.

Message Authentication Code (MAC): O MAC é calculado sobre todo o pacote SSH e um número sequencial, excluindo o campo MAC. O número sequencial é o número (32-bit) sequencial do pacote, inicializado a zero para o primeiro pacote e incrementado para cada pacote subsequente. O número sequencial não é incluído na informação enviada.



SSH (Secure Shell)

- SSH User Authentication Protocol (RFC 4252)
  - Modo do cliente ser autenticado pelo servidor
    - *Username* – identidade que o cliente alega
    - *Service name* – serviço a que o cliente quer aceder (usualmente o *SSH Connection Protocol*),
    - *Method name* – método de autenticação (*publickey*, *password* ou *hostbased*) utilizado neste pedido.

SSH User Authentication Protocol

Authenticates the client-side user to the server.

byte	SSH_MSG_USERAUTH_REQUEST (50)
string	username
string	service name
string	method name
....	method-specific fields

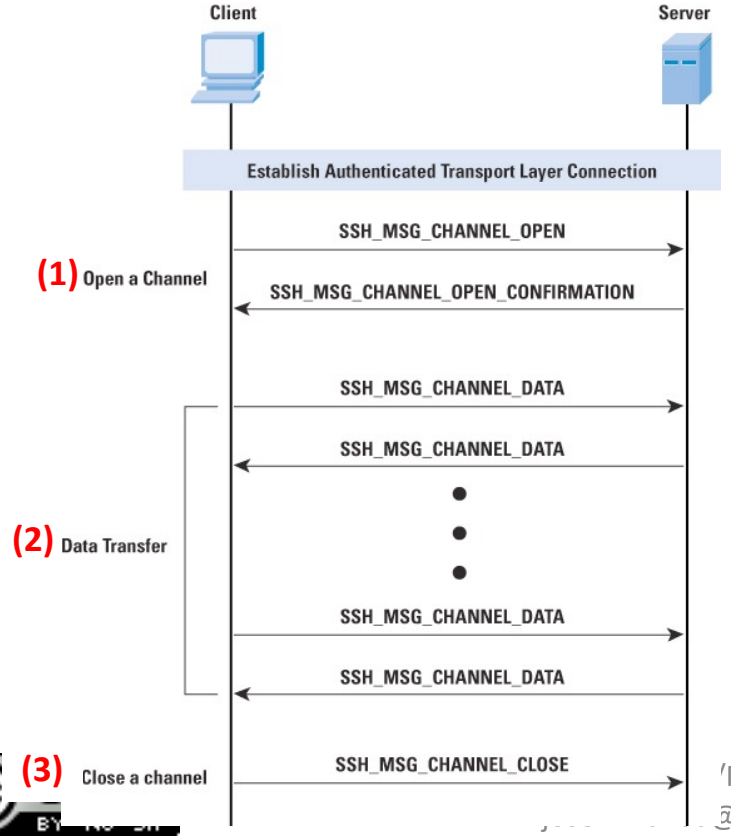
SSH (Secure Shell)

- SSH Connection Protocol (RFC 4254)

SSH Connection Protocol

Multiplexes the encrypted tunnel into several logical channels.

- Executa em cima do *SSH Transport Layer Protocol* e assume que existe uma conexão por autenticação segura (designada de túnel)
- O *Connection Protocol* permite a criação de vários canais lógicos sobre um mesmo túnel



Passo 1: A abertura de um canal, identifica o tipo de aplicação

- sessão – execução remota de um programa (e.g., shell, scp, sftp, ...),
- X11 – permite que a aplicação execute no servidor, mas a visualização gráfica seja efectuada no cliente,
- forwarded-tcpip – remote port forwarding,
- direct-tcpip – local port forwarding

para este canal e estabelece a identificação/número do canal.

Passo 2: Enquanto o canal está aberto, são transferidos dados em ambas as direcções.

Passo 3: Quando um dos lados decide fechar o canal, envia a mensagem SSH_MSG_CHANNEL_CLOSE.

SSH (Secure Shell)

- Utilização:
  - Acesso remoto:
    - Na primeira vez necessário ter atenção à fingerprint apresentada, já que a partir daí a máquina remota fica validada

```
[jepm@ProOne ~]$ ssh jepm@algo.paranoidjasmine.com
The authenticity of host 'algo.paranoidjasmine.com (163.172.150.117)' can't be established.
ECDSA key fingerprint is SHA256:hHqIkUw3XmEUnVKmFqP4ajGeFtw0P6QJnsOwEN2DZXE.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'algo.paranoidjasmine.com' (ECDSA) to the list of known hosts.
```

    - Como posso validar o RSA key fingerprint?
      - Verificar se é publicado pelo administrador do sistema remoto ou aceder fisicamente ao sistema remoto e efectuar o seguinte comando:

```
jepm@algo:~$ sudo ssh-keygen -l -f /etc/ssh/ssh_host_ecdsa_key
256 SHA256:hHqIkUw3XmEUnVKmFqP4ajGeFtw0P6QJnsOwEN2DZXE root@DF.server01 (ECDSA)
```

SSH (Secure Shell)

- Utilização:
  - Acesso remoto com autenticação do utilizador por chave pública
    - Gerar par de chaves pessoais (na máquina local) para não ser necessário introduzir password no acesso remoto (embora seja necessário introduzir a passphrase da chave privada criada):

```
jmiranda:0 ~ 1001 $ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/jmiranda/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/jmiranda/.ssh/id_rsa.
Your public key has been saved in /home/jmiranda/.ssh/id_rsa.pub.
The key fingerprint is:
56:6a:a0:7f:a3:2e:22:89:fd:9b:98:f0:05:21:a7:a0 jmiranda@clients01.
```

    - Copiar conteúdo de ~/.ssh/id_rsa.pub para a máquina remota e adicionar (na máquina remota) a ~/.ssh/authorized_keys
    - Na máquina remota efectuar

```
chmod 700 ~/.ssh
chmod 600 ~/.ssh/authorized_keys
```

de tal modo que:

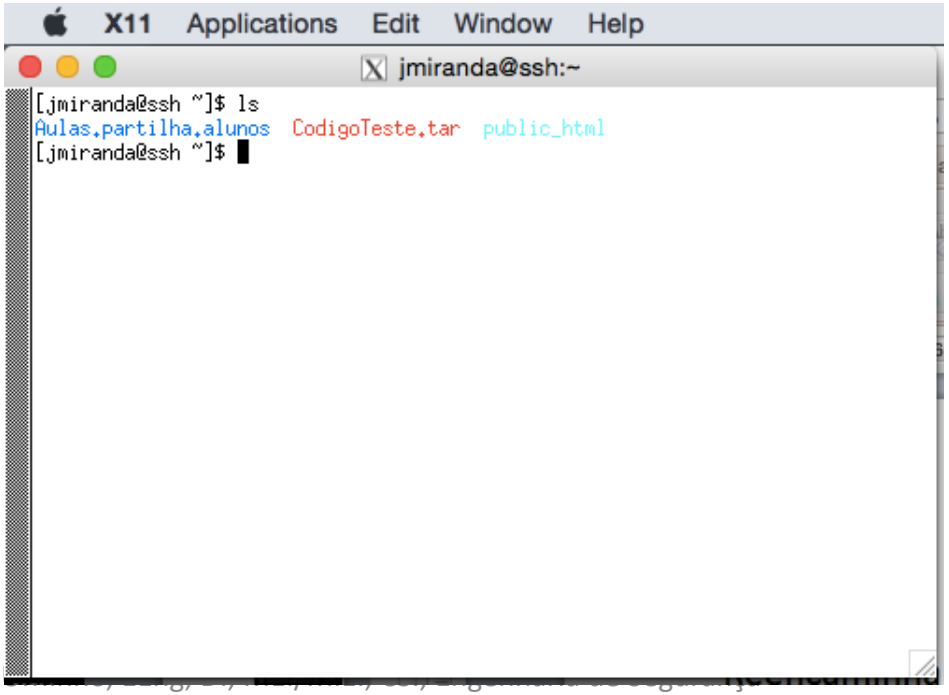
```
[jmiranda@ssh .ssh]$ ls -la
total 8
drwx------ 2 jmiranda dcc   28 Nov 26 20:29 .
drwx------ 5 jmiranda dcc 4096 Dec 15 17:19 ..
-rw-r--r-- 1 jmiranda dcc  393 Nov 26 20:29 authorized_keys
```

SSH (Secure Shell)

- Utilização:
  - Reencaminhamento de sessões X11

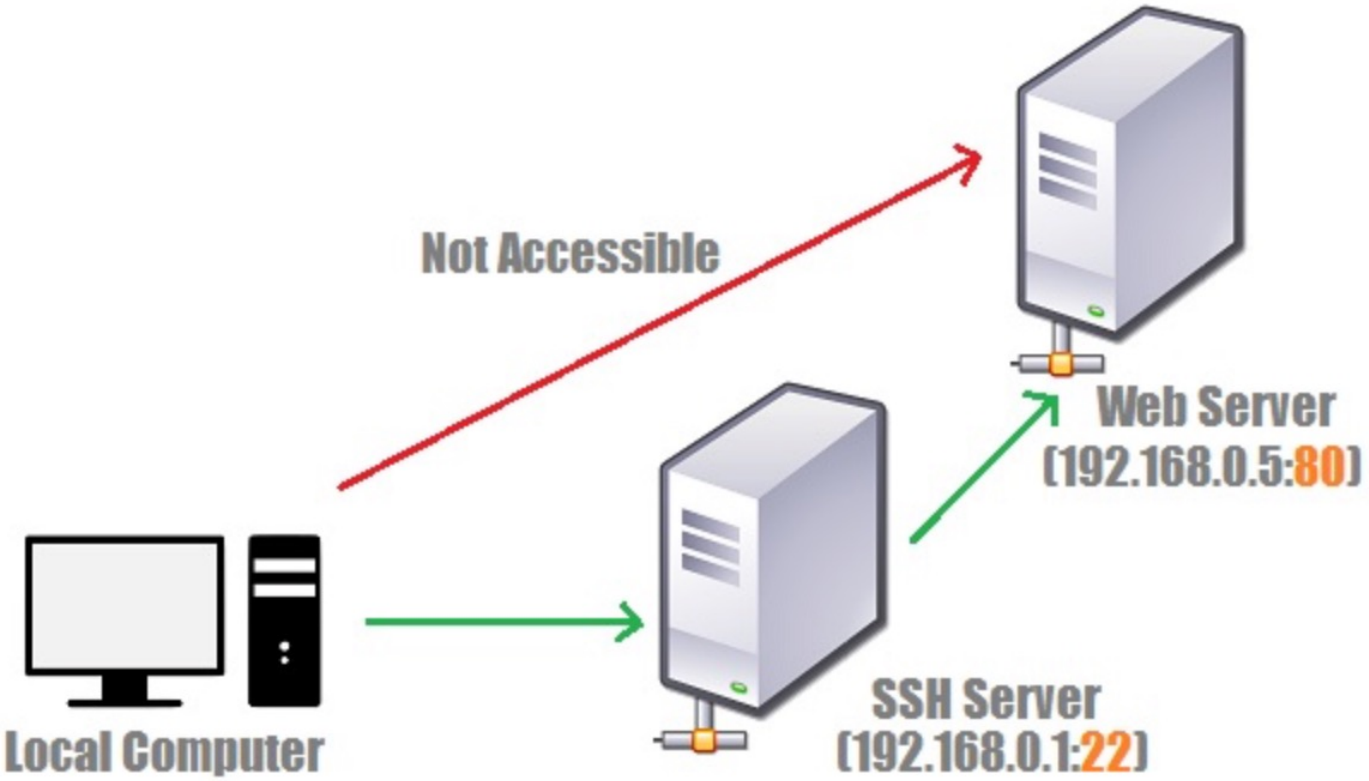
```
$ ssh -X jmiranda@ssh.alunos.dcc.fc.up.pt
```

```
[jmiranda@ssh ~]$ ls
Aulas.partilha.alunos  CodigoTeste.tar  public_html
[jmiranda@ssh ~]$ xterm&
```



SSH (Secure Shell)

- Utilização:
  - Reencaminhamento de portas TCP (*local port forward*)



SSH (Secure Shell)

- Utilização:
  - Reencaminhamento de portas TCP (*local port forward*)
    - Por exemplo, aceder ao servidor de mail remoto através de um porto local
      - Nota: -L (significa porta local) e tem o argumento

        <local-port>:<connect-to-host>:<connect-to-port>

    - Na máquina local efectuar:

```
$ ssh -L 8025:smtp.alunos.dcc.fc.up.pt:25 jmiranda@ssh.alunos.dcc.fc.up.pt
```

```
$ telnet localhost 8025
Trying ::1...
Connected to localhost.
Escape character is '^]'.
220 smtp.alunos.dcc.fc.up.pt ESMTP Postfix (2.2.8) (Mandriva Linux)
```

SSH (Secure Shell)

- Utilização:
  - Reencaminhamento invertido de portas TCP (*reverse/remote port forward*)
    - Relevante quando pretende dar acesso externo, a recursos na rede interna.
    - Por exemplo, o acesso ao porto 8888 da máquina com endereço IP 44.11.22.33 faz a ligação com o porto 8080 da máquina com endereço IP 10.0.0.3.

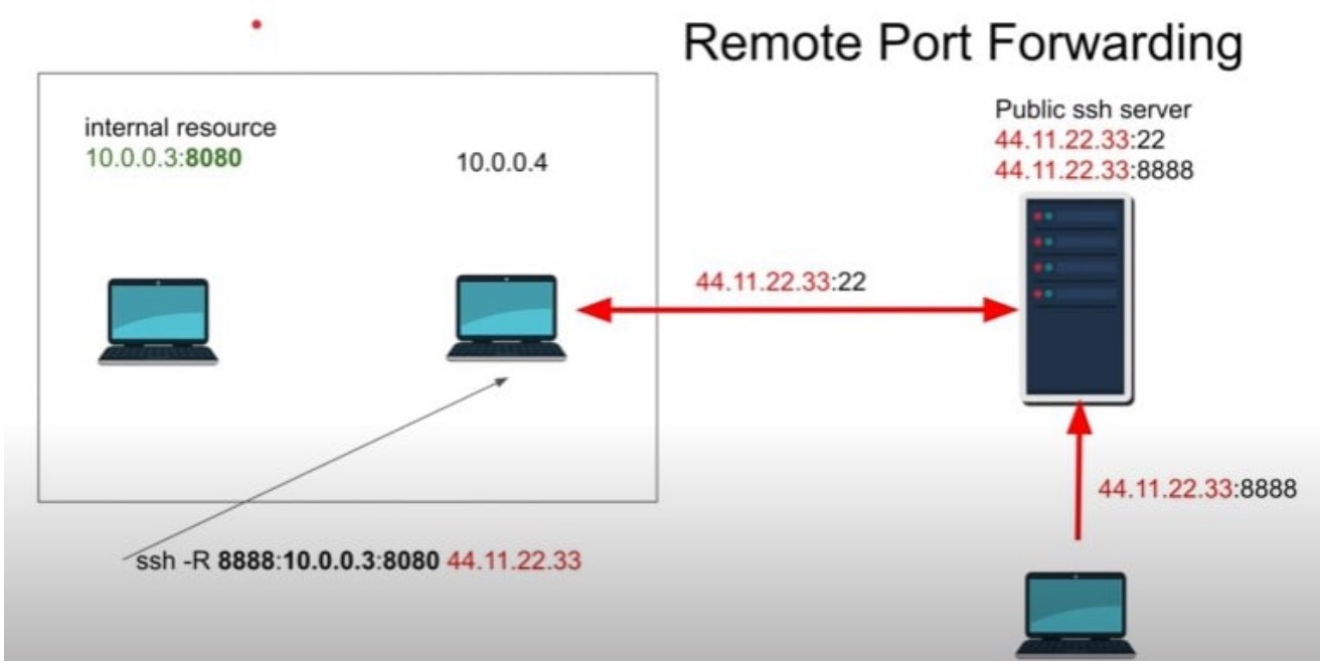


Imagem: https://www.section.io/engineering-education/ssh-tunneling/

SSH (Secure Shell)

- Utilização:
  - Reencaminhamento invertido de portas TCP (*reverse/remote port forward*)
    - Por exemplo, conectar à porta remota e começar à escuta no porto 8080 para aceder ao proxy através da máquina local e aceder à intraWeb
      - Nota: -R (significa porta remota) e tem o argumento

        <remote-port>:<connect-to-host>:<connect-to-port>

    - Na máquina local efectuar:

```
[jmiranda@ssh ~]$ ssh -R 8080:192.168.0.1:8080 jmiranda@clients.mycryptovault.com
```

    - Na máquina remota (clientes.mycryptovault.com) efectuar:

```
jmiranda:0 ~/.ssh 1019 $ curl --proxy localhost:8080 web.alunos.dcc.fc.up.pt
<html>
<body>
<!-- $Id: index.html 92365 2007-09-23 14:04:27Z oden $ -->
<!-- $HeadURL: svn+ssh://svn.mandriva.com/svn/packages/cooker/apache-conf/current/SOURCES/index.html $ --
>
<h1>It works!</h1>
</body>
</html>
```

      - De notar que a partir da máquina remota, sem reencaminhamento invertido:

```
jmiranda:0 ~/.ssh 1020 $ curl web.alunos.dcc.fc.up.pt
curl: (6) Couldn't resolve host 'web.alunos.dcc.fc.up.pt'
```

SSH Audit (ssh-audit)

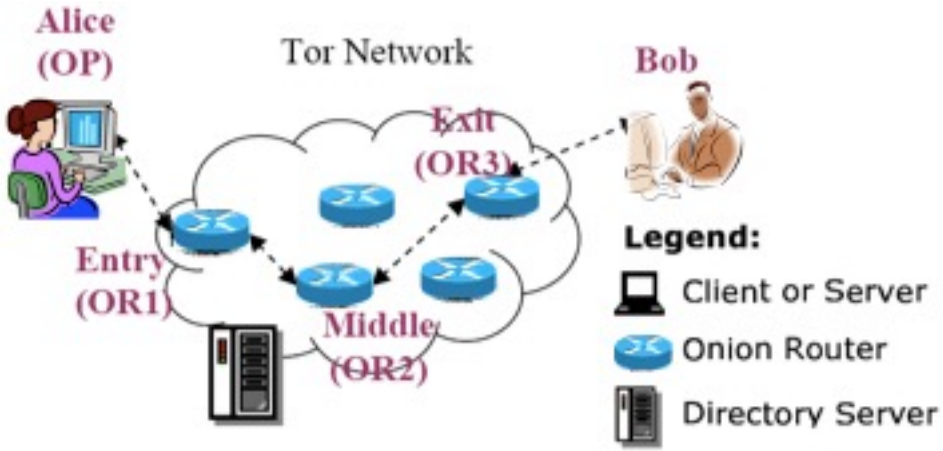
```
bash-3.2$ ./ssh-audit ssh.di.uminho.pt
# general
(gen) banner: SSH-2.0-OpenSSH_4.3
(gen) software: OpenSSH 4.3
(gen) compatibility: OpenSSH 4.2-6.6, Dropbear SSH 0.53+ (some functionality from 0.52)
(gen) compression: enabled (zlib@openssh.com)

# security
(cve) CVE-2018-15473                    -- (CVSSv2: 5.3) enumerate usernames due to timing discrepencies
(cve) CVE-2016-3115                     -- (CVSSv2: 5.5) bypass command restrictions via crafted X11 forwarding data
(cve) CVE-2014-1692                     -- (CVSSv2: 7.5) cause DoS via triggering error condition (memory corruption)
(cve) CVE-2012-0814                     -- (CVSSv2: 3.5) leak data via debug messages
(cve) CVE-2011-5000                     -- (CVSSv2: 3.5) cause DoS via large value in certain length field (memory consumption)
(cve) CVE-2010-5107                     -- (CVSSv2: 5.0) cause DoS via large number of connections (slot exhaustion)
(cve) CVE-2010-4755                     -- (CVSSv2: 4.0) cause DoS via crafted glob expression (CPU and memory consumption)
(cve) CVE-2010-4478                     -- (CVSSv2: 7.5) bypass authentication check via crafted values
(cve) CVE-2009-2904                     -- (CVSSv2: 6.9) privilege escalation via hard links to setuid programs
(cve) CVE-2008-5161                     -- (CVSSv2: 2.6) recover plaintext data from ciphertext
(cve) CVE-2008-4109                     -- (CVSSv2: 5.0) cause DoS via multiple login attempts (slot exhaustion)
(cve) CVE-2008-1657                     -- (CVSSv2: 6.5) bypass command restrictions via modifying session file
(cve) CVE-2008-1483                     -- (CVSSv2: 6.9) hijack forwarded X11 connections
(cve) CVE-2007-4752                     -- (CVSSv2: 7.5) privilege escalation via causing an X client to be trusted
(cve) CVE-2007-2243                     -- (CVSSv2: 5.0) discover valid usernames through different responses
(cve) CVE-2006-5052                     -- (CVSSv2: 5.0) discover valid usernames through different responses
(cve) CVE-2006-5051                     -- (CVSSv2: 9.3) cause DoS or execute arbitrary code (double free)
(cve) CVE-2006-4924                     -- (CVSSv2: 7.8) cause DoS via crafted packet (CPU consumption)

# encryption algorithms (ciphers)
(enc) aes128-ctr                        -- [info] available since OpenSSH 3.7, Dropbear SSH 0.52
(enc) aes192-ctr                        -- [info] available since OpenSSH 3.7
(enc) aes256-ctr                        -- [info] available since OpenSSH 3.7, Dropbear SSH 0.52
(enc) arcfour256                        -- [fail] removed (in server) since OpenSSH 6.7, unsafe algorithm

# message authentication code algorithms
(mac) hmac-md5                          -- [fail] removed (in server) since OpenSSH 6.7, unsafe algorithm
                                        `- [warn] disabled (in client) since OpenSSH 7.2, legacy algorithm

# algorithm recommendations (for OpenSSH 4.3)
(rec) -3des-cbc                         -- enc algorithm to remove
(rec) -aes128-cbc                       -- enc algorithm to remove
(rec) -aes192-cbc                       -- enc algorithm to remove
(rec) -aes256-cbc                       -- enc algorithm to remove
(rec) -arcfour                          -- enc algorithm to remove
(rec) -arcfour128                       -- enc algorithm to remove

# additional info
(nfo) For hardening guides on common OSes, please see: <https://www.ssh-audit.com/hardening_guides.html>
```

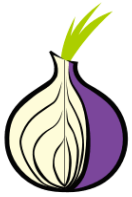
Tópicos

- **Parte X: Aplicações e protocolos**
  - TLS
  - SSH
  - **TOR**
  - *Blockchain*/DLT

TOR (The Onion Router)

- **Protocolo criptográfico de rede** cujo objectivo é:
  - Garantir a **anonimidade ponto-a-ponto** (ao nível não aplicacional) de um utilizador na Internet;
    - Note que todos os protocolos que vimos até agora estabelecem túneis seguros/privados/confidenciais, mas não permitem anonimidade ponto-a-ponto (ao nível não aplicacional), na medida em que os *headers* dos pacotes (TCP / UDP / IP) ainda revelam muita informação sobre o utilizador.
  - Permitir disponibilizar **serviços anónimos** (*hidden services*) – www e outros serviços – sem revelar a localização dos mesmos.





TOR (The Onion Router)

- Rede sobreposta à Internet constituída por ***Onion Routers*** (**OR**)
  - Cada OR executa como um processo normal do utilizador sem necessidade de privilégios especiais;
  - Cada OR conecta-se a outros OR através de uma conexão TLS;
  - Existem OR "mais confiáveis" que actuam como *Directory Server* – fornecem listas assinadas dos OR conhecidos e seu estado actual, que são descarregadas periodicamente pelos utilizadores do TOR –;
  - Cada OR tem um par de chaves de identidade de longo prazo (*identity key*) utilizado para assinar os certificados TLS, o descritor do OR (contém chaves públicas, endereço, largura de banda, política de saída, etc.) e a directoria (no caso dos *Directory Server*);
  - Cada OR tem um par de chaves de curto prazo (*onion key*), rodadas periodicamente, utilizado para estabelecer chaves de sessão com o utilizador (através de Diffie-Hellman).
- Cada utilizador executa um software local: **Onion Proxy** (**OP**)
  - OP obtém dados da directoria, estabelece circuitos através da rede TOR e gere conexões das aplicações do utilizador.

How Tor Works:

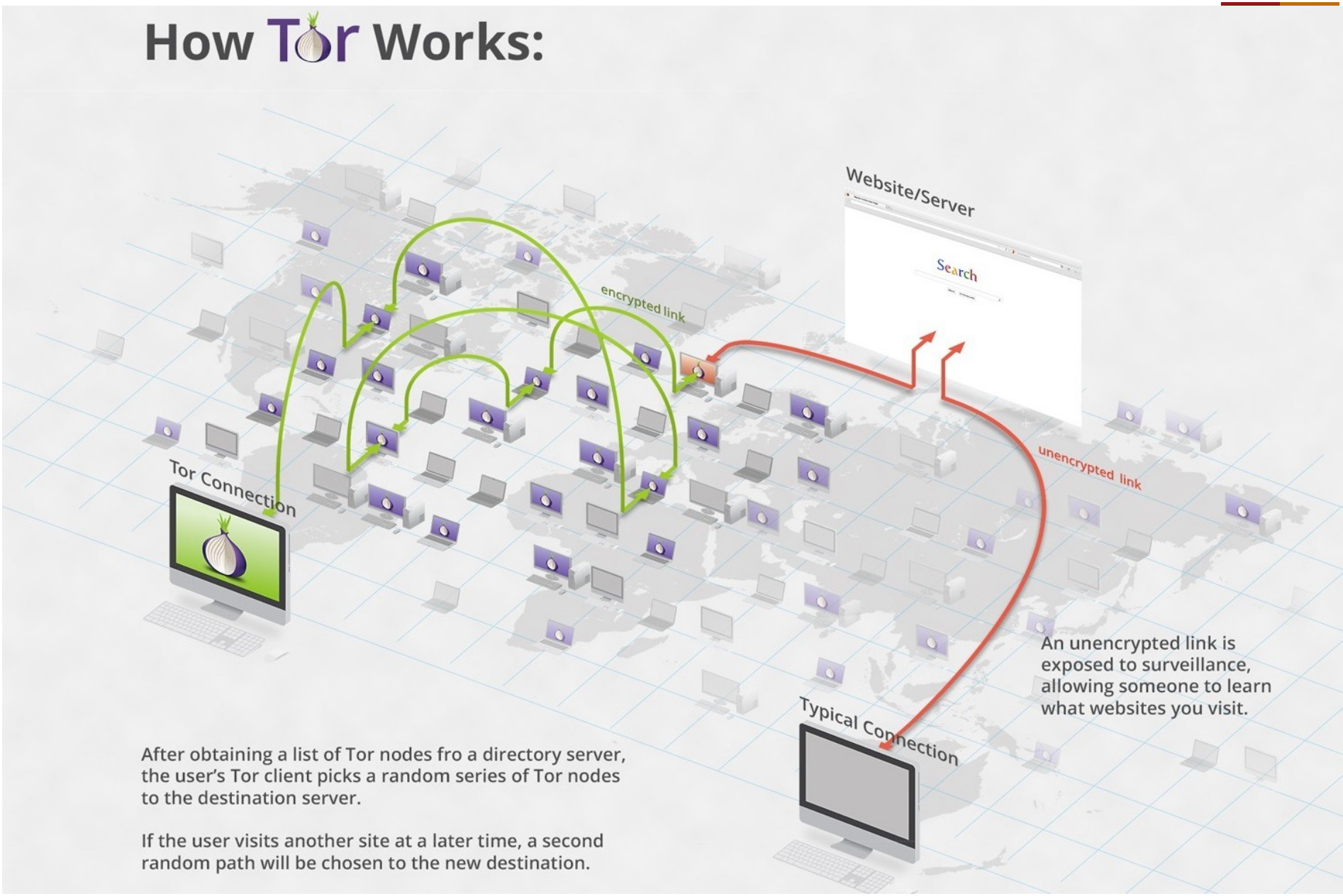


Imagem: https://www.extremetech.com/internet/226106-onionscan-tests-dark-web-sites-to-see-if-they-really-are-anonymous

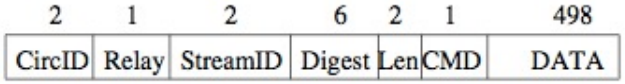
TOR (The Onion Router)

- OR comunicam entre si e com OP através de conexões TLS em pacotes (células) de tamanho fixo.
  - Cada célula tem 512 bytes e é constituída por um *header* e um *payload*;
  - O *header* inclui identificador do circuito circID que indica a que circuito a célula se refere (vários circuitos podem ser multiplexados sobre a mesma ligação TLS) e um comando CMD que descreve o que fazer com o *payload*;

  - Uma célula pode ser uma *control cell* (sempre interpretada pelo OR que a recebe) ou uma *relay cell* (leva dados ponto a ponto);
  - O CMD das *control cells* pode ser:
    - padding (utilizado para *keepalive*),
    - create/created (para criar novo circuito), ou
    - destroy (para finalizar circuito);

TOR (The Onion Router)

- OR comunicam entre si e com OP através de conexões TLS em pacotes (células) de tamanho fixo.
  - As *relay cells* têm um *header* adicional com o streamID (identificador do stream: vários streams podem ser multiplexados sobre o mesmo circuito), hash ponto-a-ponto (para verificação de integridade), o tamanho do payload do relay e o CMD do relay;



  - Todo o conteúdo do relay header e relay payload (i.e., o payload da célula) é cifrado ou decifrado (AES 128 bits) sequencialmente à medida que a célula se move ao longo do circuito;
  - O CMD do relay pode ser:
    - data (para dados a serem comunicados na stream),
    - begin (para abrir nova stream), end (para fechar stream),
    - teardown (para fechar uma stream "estragada"),
    - connected (para notificar o OP que foi efectuado um begin com sucesso),
    - extend/extended (para extender o circuito por mais um OR),
    - truncate/truncated (para destruir apenas parte do circuito),
    - sendme (para controlo de congestionamento num OR), ou
    - drop (para testar stream);

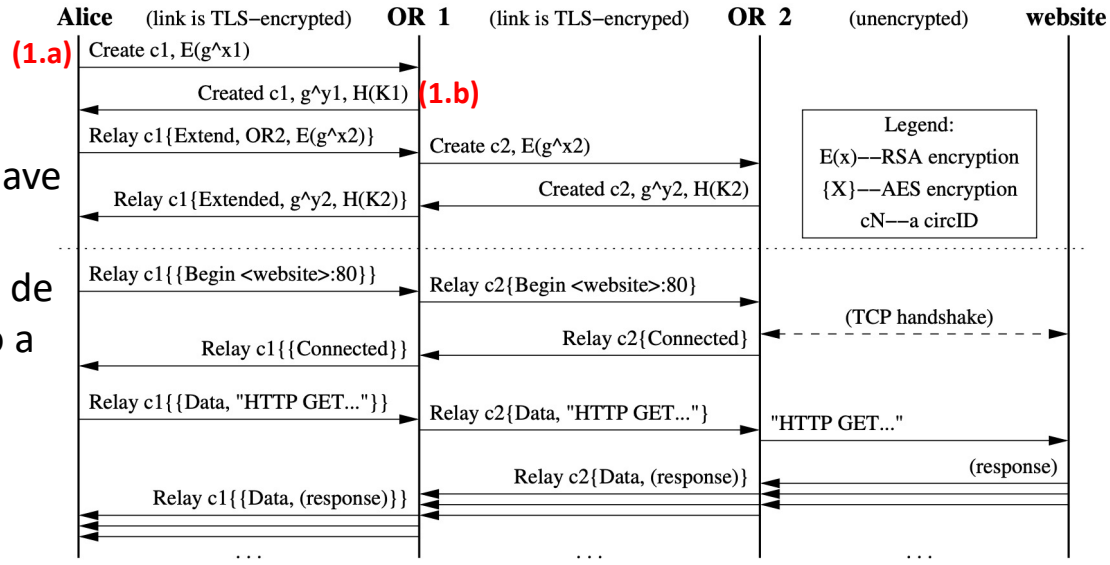
TOR - Anonimização

- OP pré-estabelece circuitos (normalmente de 3 OR) e muda para um novo circuito a cada 10 minutos, garantindo que apenas um número limitado de pedidos podem ser ligados uns aos outros no OR de saída;
- OP constrói o circuito incrementalmente, negociando uma chave simétrica com cada OR do circuito, OR a OR. Note-se que escolhe os OR a partir da lista de OR fornecida pelo *Directory Server* que tem associado as chaves de longo termo (*identity key* para assinar certificados TLS) e de curto termo (*onion key* para estabelecer chaves de sessão por Diffie-Hellman).

Passo 1a: O OP (Alice) envia uma célula de controlo *create* para o primeiro nodo (OR1) do circuito escolhido pelo OP, escolhendo um novo circID e com o payload da célula contendo a primeira metade da troca de chaves Diffie-Hellman (g^{x1} cifrado com a chave pública da *onion key* de OR1).

Passo 1b: O OR1 responde com uma célula de controlo *created*, contendo g^{y1} assim como a hash da chave K_1 negociada ($K_1 = g^{x1.y1}$).

A partir deste momento, OP e OR1 podem comunicar a célula de *relay* com o payload cifrado com a chave K_1.



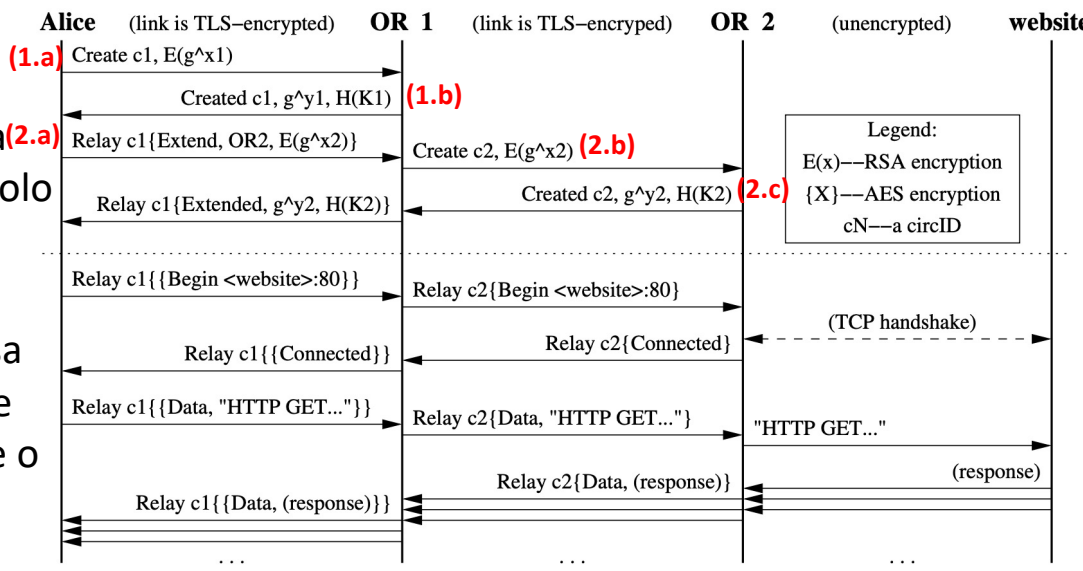
TOR - Anonimização

Passo 2a: Para extender o circuito, o OP (Alice) envia uma célula de *relay extend* ao OR1, identificando o próximo OR (OR2) e com g^{x2} cifrado com a chave pública da *onion key* de OR2 ($E(g^{x2})$).

Passo 2b: OR1 escolhe um novo CircID, copia $E(g^{x2})$ para o payload de uma célula de controlo *create* e, envia-a a OR2.

Passo 2c: OR2 responde com uma célula de controlo *created* e OR1 copia o payload dessa célula para uma célula de *relay extended* que envia a OP. O circuito está extendido a OR2 e o OP e OR2 partilham a chave comum ($K_2 = g^{x2.y2}$).

A partir deste momento, OP e OR2 podem comunicar a célula de *relay* com o payload cifrado com a chave K_2.

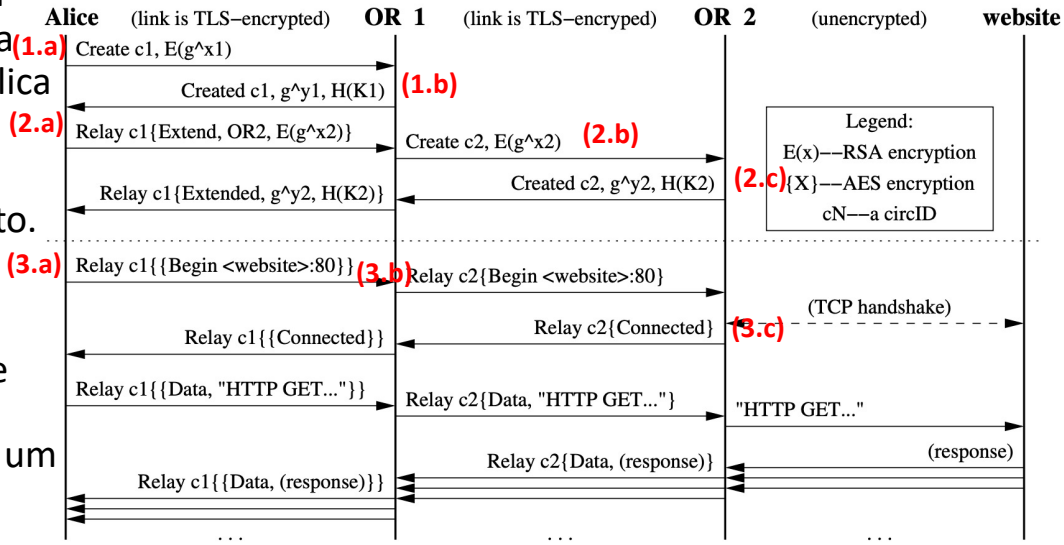


Para extender o circuito para nodos adicionais (OR_{n+1}), OP (Alice) efetua os passos anteriores, indicando sempre ao último OR (OR_n) no circuito para extender ao novo OR (OR_{n+1}).

TOR - Anonimização

O Protocolo de estabelecimento do circuito garante autenticação unilateral (OP sabe que está a trocar chaves com o OR, mas o OR não sabe quem está a abrir o circuito – i.e., OP não usa a sua chave pública e mantém-se anónimo).

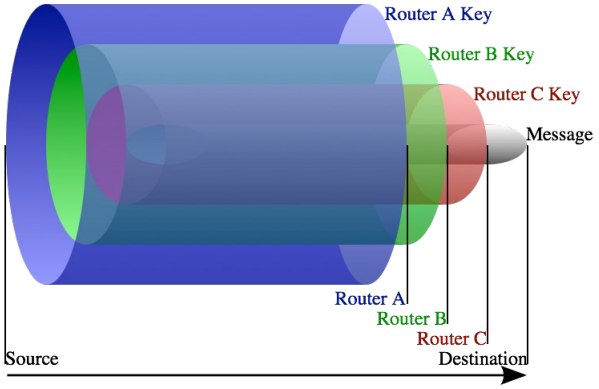
Assim que o circuito está estabelecido, OP pode enviar células de *relay* até ao último OR do circuito.



Passo 3a: OP (Alice) envia a célula de *relay* ao OR destinatário. Para construir essa célula, insere-lhe todos os dados necessários, gera o hash/digest e cifra com cada chave simétrica trocada com cada um dos OR (neste caso cifra com OR1 e depois com OR2).

Passo 3b: o primeiro OR (OR1) decifra o payload e verifica se o hash está correcto. Se estiver, efectua o comando pedido pelo OP. Se não estiver, pega no payload decifrado e envia uma célula de *relay* para o OR seguinte no circuito.

Condição de paragem, i.e., chegou ao OR de destino

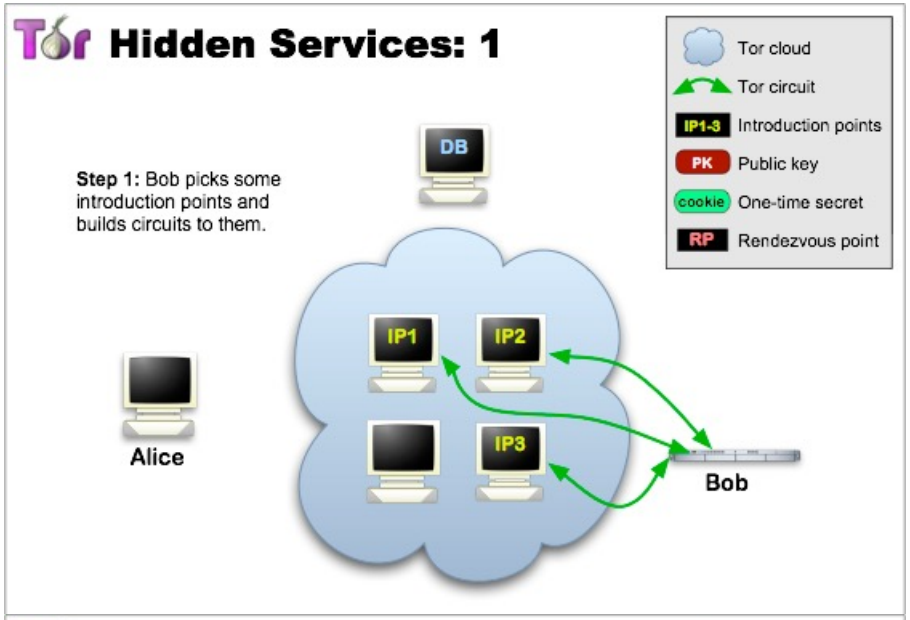


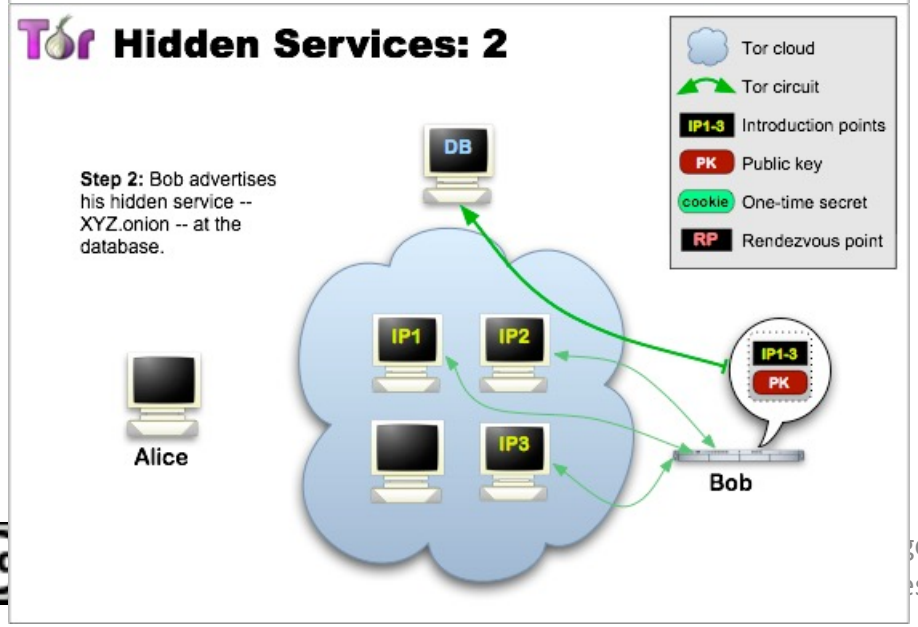
Passo 3c: O OR final responde, através do circuito estabelecido, com uma célula de *relay* com o payload cifrado para o OP. Os OR intermédios adicionam novos níveis de cifra à medida que a "reencaminham" para o OP.

TOR – Pontos de *Rendezvous* e serviços anónimos

- **Pontos de *Rendezvous*** são o suporte para a disponibilização de serviços anónimos (também designados por *responder anonymity*).
  - Na rede TOR, a disponibilização de **serviços anónimos** permite a um OP (Bob) disponibilizar serviços TCP (por exemplo, servidor *web*) sem revelar o seu endereço IP.
  - Como o OP (Alice) que acede ao serviço anónimo também é anonimizado, tanto o OP que acede como o OP que é acedido são anónimos.

TOR – Pontos de *Rendezvous* e serviços anónimos



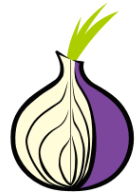


<u>Passos 1 e 2</u>:

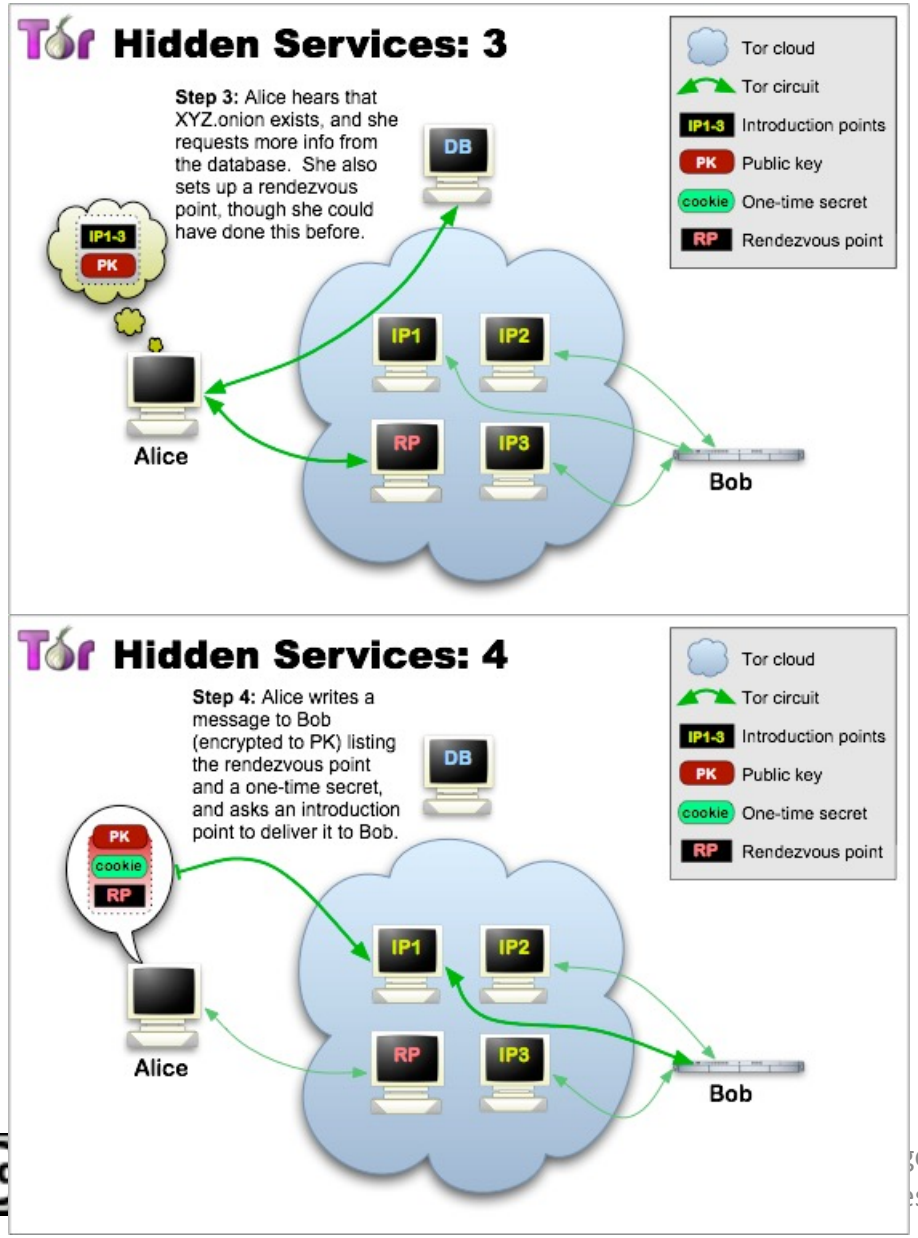
a. Bob gera um par de chaves de longo prazo para identificar o seu serviço Web (a chave pública passa a ser o identificador do serviço);

b. Bob escolhe alguns pontos de introdução (*introduction points*) e anuncia-os no *Directory Server*, assinando o anuncio (descritor do serviço) com a sua chave privada;
   - O descritor do serviço anónimo contém a chave pública do serviço e um sumário de cada ponto de introdução;
   - O descritor/serviço será encontrado pelos clientes que acederem a XYZ.onion na rede TOR, onde XYZ é um nome de 16 caracteres derivado da chave pública do serviço.

c. Bob cria um circuito TOR (conforme visto nos últimos slides) para cada um dos pontos de introdução e pede-lhes para esperarem por pedidos.



TOR – Pontos de Rendezvous e serviços anónimos



Passos 3 e 4:

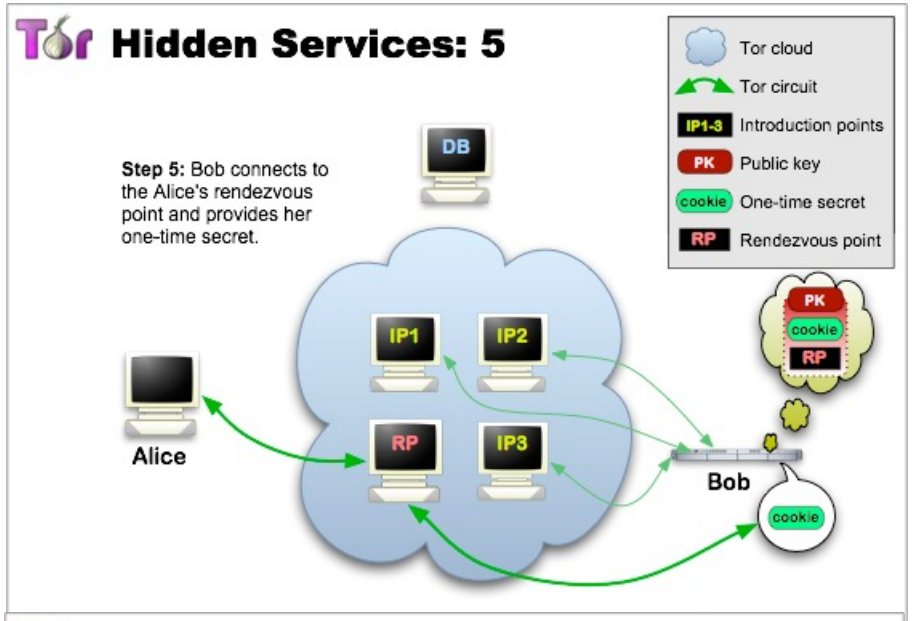
a. Alice sabe da existência do serviço XYZ.onion e acede aos seus detalhes (chave pública e *introduction points*) no TOR através do Directory Server;

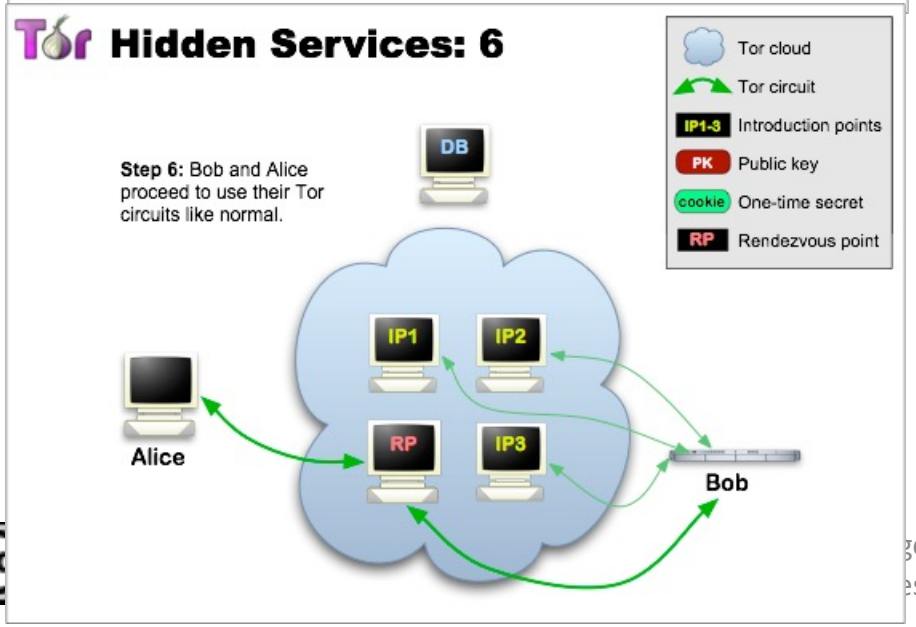
b. Alice escolhe um OR como ponto de *rendezvous* (RP) para a conexão com o serviço XYZ.onion;

c. Alice constrói um circuito TOR até ao RP e fornece-lhe um "*rendezvous cookie*" (segredo aleatório único) para posterior reconhecimento do serviço XYZ.onion;

d. Alice abre um *stream* anónimo até um dos pontos de introdução e fornece-lhe uma mensagem (cifrada com a chave pública de XYZ.onion) com informação sobre o RP, o "*rendezvous cookie*" e o inicio de troca de chaves Diffie-Hellman. O ponto de introdução reencaminha a mensagem para o serviço XYZ.onion através do circuito TOR criado nos passos anteriores.

TOR – Pontos de Rendezvous e serviços anónimos





Passos 5 e 6:

a. Se Bob (XYZ.onion) pretender falar com Alice (OP), Bob cria um circuito TOR até ao RP e envia o "*rendezvous cookie*", a segunda parte da troca de chaves Diffie-Hellman e um *hash* da chave de sessão que agora partilha com Alice;

b. O RP conecta o circuito de Alice com o circuito de Bob (normalmente o circuito consiste de 6 OR: 3 escolhidos por Alice sendo o terceiro o RP e, outros 3 escolhidos por Bob). Note-se que o RP não consegue reconhecer Alice, Bob nem os dados que transmitem;

c. Alice envia uma célula de *relay begin* através do circuito que, ao chegar ao OP de Bob, conecta com o serviço disponibilizado por Bob (por exemplo, um servidor Web);

d. Um *stream* anónimo foi estabelecido e Alice e Bob comunicam da forma normal num *stream* TOR.

TOR – Possíveis Ataques

O que acontece se o nosso servidor actuar como nodo de entrada?

- Sabe que o IP de John está a utilizar a rede TOR (interessante para SIGINT – *signal intelligence* – mas pouco mais).

O que acontece se o nosso servidor actuar como nodo intermédio?

- Nada. Recebe informação cifrada de um nodo TOR que vai para outro nodo TOR.

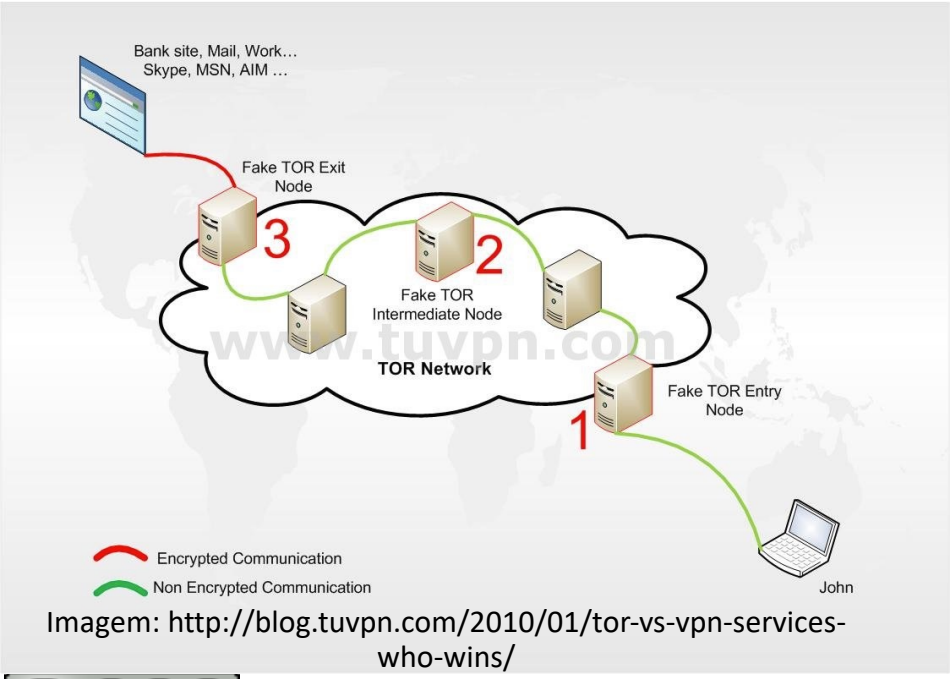


Imagem: http://blog.tuvpn.com/2010/01/tor-vs-vpn-services-who-wins/

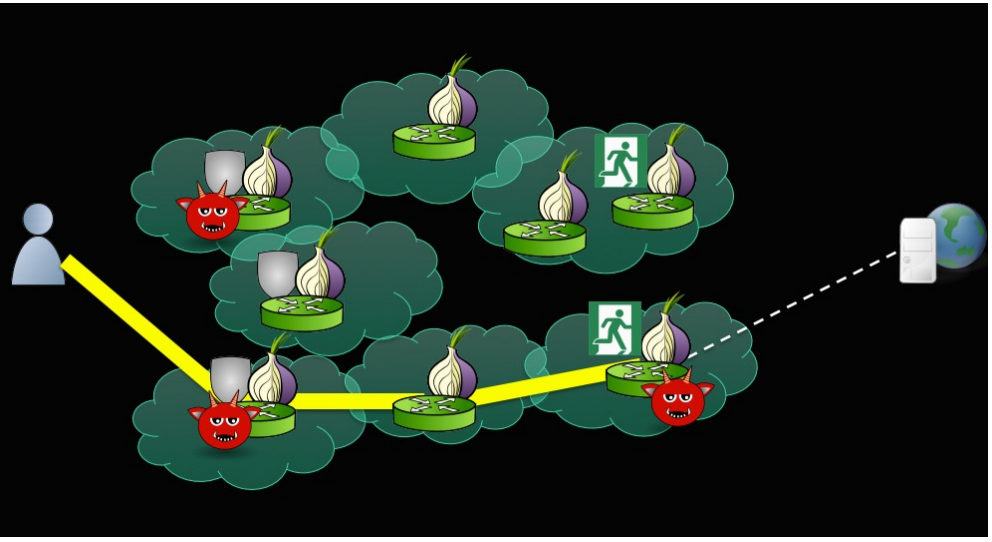
O que acontece se o nosso servidor actuar como nodo de saída?

- A missão do nodo de saída é decifrar a comunicação, e enviá-la para o serviço (web) de destino, pelo que não sabe quem foi o originante da comunicação mas sabe qual é a comunicação.
- Podemos argumentar que se o destino é SSL/TLS, não há hipótese de aceder à comunicação em claro do John.
- Errado. Foi demonstrado que é possível um *MitM attack* sobre o SSL/TLS, no nodo de saída.

TOR – Possíveis Ataques

O que acontece se um servidor nosso actuar como nodo de entrada e outro servidor nosso actuar como nodo de saída?

- É possível calcular coeficientes de correlação – através de fórmulas matemáticas da área da probabilidade e estatística – na análise de pacotes nos dois nodos, baseado na frequência, timing e tamanho do pacote.
- Supõe-se que 80% dos utilizadores podem ser de-anonimizados no período de 6 meses.



- Custo elevado, já que uma grande percentagem dos OR (TOR tem cerca de 7.000 OR) têm que pertencer à entidade que queira efetuar um ataque de correlação.
- Utilizado por agências governamentais (BND, GCHQ, NSA).
- Supõe-se que o "Silk Road 2.0" foi de-anonimizado com base neste ataque.

Imagem: https://www.deepdotweb.com/2016/10/25/tors-biggest-threat-correlation-attack/

TOR

- Exemplo efectuado em menos de dez minutos, a partir da mesma máquina, no site www.whatismyip.com

Tor Browser

IP Address: 87.106.148.90
City: Karlsruhe
State/Region: Baden-wurttemberg
Country Code: DE
Postal Code: 76229
ISP: 1&1 Internet Ag
Time Zone: +01:00 UTC/GMT
Latitude: 49.0047
Longitude: 8.3858

IP Address: 92.222.172.41
City: Roubaix
State/Region: Nord-pas-de-calais
Country Code: FR
Postal Code: 59689
ISP: Ovh Sas
Time Zone: +01:00 UTC/GMT
Latitude: 50.6942
Longitude: 3.1746

IP Address: 94.242.206.170
City: Steinsel
State/Region: Luxembourg
Country Code: LU
Postal Code: I-7349
ISP: Root Sa
Time Zone: +01:00 UTC/GMT
Latitude: 49.6769
Longitude: 6.1239

Firefox

IP Address: 94.132.113.217
City: Porto
State/Region: Porto
Country Code: PT
Postal Code: -
ISP: TVCABO Portugal, S.A.
Latitude: 41.1496
Longitude: -8.611

- Qual a explicação?

TOR

Bibliography (English):

- TOR overview - https://www.torproject.org/about/overview.html.en
- Tor: The Second-Generation Onion Router - https://svn.torproject.org/svn/projects/design-paper/tor-design.pdf

Tópicos

- **Parte X: Aplicações e protocolos**
  - TLS
  - SSH
  - TOR
  - ***Blockchain*/DLT**

Blockchain / DLT (Distributed Ledger Technology)

Blockchains

- **tamper evident** and **tamper resistant digital ledgers**
- **implemented in a distributed fashion** (i.e., without a central repository) and
- **usually without a central authority** (i.e., a bank, company, or government).
- At their basic level, they
  - enable a community of users to **record transactions in a shared ledger** within that community, such that
  - under normal operation of the blockchain network **no transaction can be changed once published**.

Blockchain

In 2008, the blockchain idea was combined with several other technologies and computing concepts to create **modern cryptocurrencies**: **electronic cash protected through cryptographic mechanisms** instead of a **central repository or authority**.

The first such blockchain based cryptocurrency was **Bitcoin**.

Within the Bitcoin blockchain:

- Information representing electronic cash is attached to a digital address (users use **pseudonymous** – meaning that users are anonymous, but their account identifiers are not).
- Bitcoin users can digitally sign and transfer rights to that information to another user and the Bitcoin blockchain records this transfer publicly, allowing all participants of the network to independently verify the validity of the transactions (all **transactions** are **publicly visible**).

The Bitcoin blockchain is stored, maintained, and collaboratively managed by a distributed group of participants (**new cryptocurrency** is **issued** to those users who manage to publish new blocks and maintain copies of the ledger; such users are called ***miners***). This, along with certain cryptographic mechanisms, makes the **blockchain resilient to attempts to alter the ledger** later (modifying blocks or forging transactions).

Blockchain

Blockchain technology is the foundation of modern **cryptocurrencies**, so named because of the **heavy usage of cryptographic functions**.

- Users utilize **public and private keys** to **digitally sign** and **securely transact** within the system.
- For cryptocurrency based blockchain networks which utilize **mining**, users may **solve puzzles using cryptographic hash functions** in hopes of being rewarded with a fixed amount of the cryptocurrency.

However, blockchain technology may be more broadly applicable than cryptocurrencies; there is a growing interest in other sectors.

Organizations considering implementing blockchain technology need to understand fundamental aspects of the technology – a **blockchain is just one part of a solution**.

Blockchain

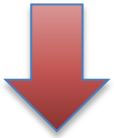
Blockchain networks common core concepts:

- **Pseudo-anonymity** because accounts can be created without any identification or authorization process;
- Blockchains are a **distributed digital ledgers** of **cryptographically signed transactions** that are grouped into **blocks**;
- Unlike traditional databases, **transactions and values** in a blockchain are **not overridden**;
- The ledger is **shared** amongst multiple participants;
- Blockchain can be **distributed**, making it more **resilient to attacks by bad actors** (increasing the number of nodes, the ability for a bad actor to impact the consensus protocol used by the blockchain is reduced);
- Each **transaction** involves one or more blockchain network users and a recording of what happened, and it is **digitally signed** by the user who submitted the transaction.
- Each **block** is comprised of:
  - **block header** containing metadata about the block, and
  - **block data** containing a set of transactions and other related data.
- Each block (except for the very first block of the blockchain) is **cryptographically linked** to the previous one (making it tamper evident) after validation and undergoing a **consensus decision**.
- New blocks are **replicated** across copies of the ledger within the network, and any conflicts are resolved automatically using established rules.
- As new blocks are added, older blocks become more difficult to modify (creating tamper resistance).

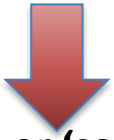
Blockchain

There are two general high-level categories for blockchain:

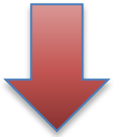
- **Permissionless blockchain** - decentralized ledger platforms open to anyone reading and publishing blocks, without needing permission from any authority



Malicious users may attempt to publish blocks in a way that subverts the system



To prevent this, a multiparty agreement or **'consensus' system** (proof of work or proof of stake) that requires users to expend or maintain resources when attempting to publish blocks is used

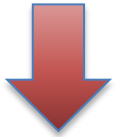


Usually, to promote non-malicious behavior, **rewards** the publishers of protocol-conforming blocks with a native cryptocurrency

Blockchain

There are two general high-level categories for blockchain:

- **Permissionless blockchain** - decentralized ledger platforms open to anyone reading and publishing blocks, without needing permission from any authority
- **Permissioned blockchain** - limit participation to specific people or organizations and allow finer-grained controls.
  - publishing blocks must be authorized by some authority
  - possible to restrict read access and to restrict who can issue transactions
  - use consensus models for publishing blocks, but these methods often do not require the expense or maintenance of resources (because those maintaining the blockchain have a level of trust with each other, since they were all authorized to publish blocks and since their authorization can be revoked if they misbehave)



Consensus models in permissioned blockchain networks are then usually faster and less computationally expensive.

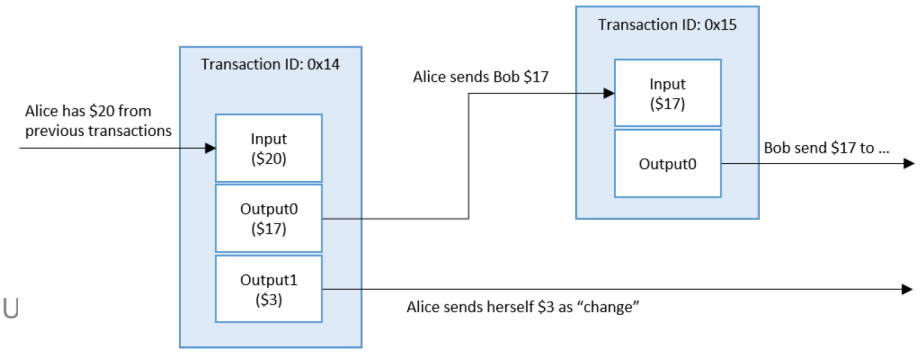
Blockchain - Components

- **Cryptographic primitives**
  - Cryptographic Hash Functions (SHA256, Keccak/SHA-3, RIPEMD-160) for:
    - Address derivation;
    - Creating unique identifiers;
    - Securing the block data – a publishing node will hash the block data, creating a digest that will be stored within the block header;
    - Securing the block header – the current block header's hash digest will be included within the next block's header, where it will secure the current block header data. Because the block header includes a hash representation of the block data, the block data itself is also secured when the block header digest is stored in the next block.
  - Cryptographic Nonce
    - Arbitrary number that is only used once;
    - combined with data to produce different hash digests per nonce:

      hash (data + nonce) = digest
    - technique utilized in the proof of work consensus model

Blockchain - Components

- **Transactions**
  - Represent an interaction between parties. For example:
    - transfer of the cryptocurrency between blockchain network users, or
    - post data on the blockchain / record activities occurring on digital or physical assets
      - In the case of smart contract systems, transactions can be used to send data, process that data, and store some result on the blockchain.
  - Blockchain mechanism for transacting is largely the same:
    - blockchain network user sends information to the blockchain network;
    - information sent may include the sender's address (or another relevant identifier), sender's public key, a digital signature (for determining the validity and authenticity of a transaction), transaction inputs and transaction outputs.



Example Cryptocurrency Transaction

Blockchain - Components

- **Asymmetric / Public key cryptography**
  - Enables a trust relationship between users who do not know or trust one another, by providing a mechanism to verify the integrity and authenticity of transactions while at the same time allowing transactions to remain public ➔ transactions are 'digitally signed';
  - Use of public key cryptography (ECDSA) in many blockchain networks:
    - Private keys are used to digitally sign transactions.
    - Public keys are used to derive addresses.
    - Public keys are used to verify signatures generated with private keys.
    - Public key cryptography provides the ability to verify that the user transferring value (money, data, …) to another user is in possession of the private key capable of signing the transaction.

Blockchain - Components

- **Addresses and Address Derivation**
  - Most blockchain implementations make use of addresses as the "to" and "from" endpoints in a transaction;
  - An ***address*** is a short, alphanumeric string of characters derived from the blockchain network user's public key using a cryptographic hash function.

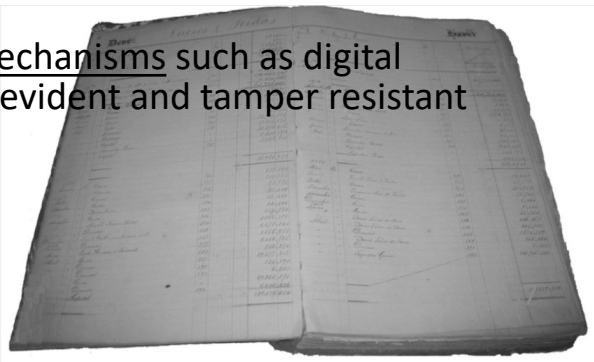
public key → cryptographic hash function → address

Blockchain - Components

- **Private Key Storage**
  - Users must manage and securely store their own private keys, using software/hardware (***wallet***);
  - The *wallet* can store private keys, public keys, and associated addresses. It may also perform other functions, such as calculating the total number of digital assets a user may have.
  - If a user loses a private key, then any digital asset associated with that key is lost, because it is computationally infeasible to regenerate the same private key.
  - If a private key is stolen, the attacker will have full access to all digital assets controlled by that private key.
  - Private key storage is an extremely important aspect of blockchain technology.

Blockchain - Components

- **Ledger (DLT - Distributed Ledger Technology)**
  - Collection of transactions;
  - Blockchain technology enables both distributed ownership as well as a distributed physical architecture of the ledger:
    - [**Trust**] Blockchain network is distributed by design, creating many backup copies all updating and syncing to the same ledger data between peers. A key benefit to blockchain technology is that every user can maintain their own copy of the ledger.
    - [**Security**] Blockchain network is a heterogeneous network, where the software, hardware and network infrastructure are all different. Because of the many differences between nodes on the blockchain network, an attack on one node is not guaranteed to work on other nodes.
    - [**Resilience**] Blockchain network can be comprised of geographically diverse nodes which may be found around the world. Because of this, and the blockchain network working in a peer-to-peer fashion, it is resilient to the loss of any node, or even an entire region of nodes.
    - [**Reliability**] Blockchain network must check that all transactions are valid; if a malicious node was transmitting invalid transactions, others would detect and ignore them, preventing the invalid transactions from propagating throughout the blockchain network.
    - [**Complete**] Blockchain network holds all accepted transactions within its distributed ledger. To build a new block, a reference must be made to a previous block – therefore building on top of it. If a publishing node did not include a reference to the latest block, other nodes would reject it.
    - [**Tamper resistant**] Blockchain network utilizes cryptographic mechanisms such as digital signatures and cryptographic hash functions to provide tamper evident and tamper resistant ledgers .



jose.miranda@devisefutures.com

Blockchain - Components

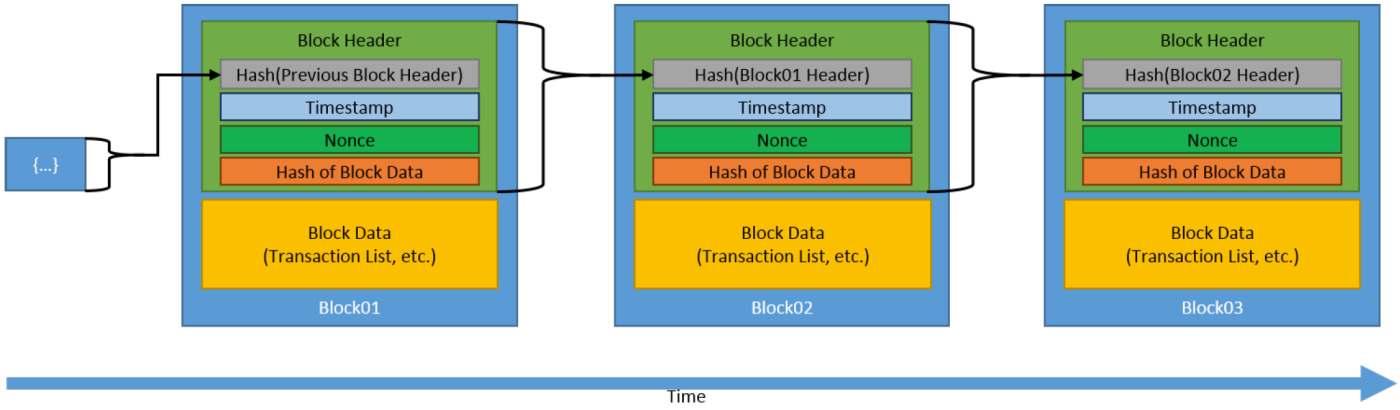
- **Blocks**
  - Blockchain network **users** submit **candidate transactions** to the blockchain network via software (desktop applications, smartphone applications, digital wallets, web services, etc.);
  - The software sends these transactions to a node or **nodes** within the blockchain network;
  - Once a **pending transaction** has been distributed to nodes, it must then wait in a queue until it is added to the blockchain by a publishing node;
  - Transactions are added to the blockchain when a **publishing node** publishes a block;
  - A ***block*** contains:
    - *block header* (metadata for this block) and
    - *block data* (list of validated and authentic transactions which have been submitted to the blockchain network).
      - Validity and authenticity is ensured by checking that the transaction is correctly formatted and that the providers of digital assets in each transaction (listed in the transaction's 'input' values) have each cryptographically signed the transaction.

Blockchain - Components

- **Blocks**
  - **Block header** usually contains:
    - Block number (also known as block height in some blockchain networks).
    - Previous block header's hash value.
    - Hash representation of the block data (different methods can be used to accomplish this, such as a generating a Merkle tree, and storing the root hash, or by utilizing a hash of all the combined block data).
    - Timestamp.
    - Size of the block.
    - Nonce value (for blockchain networks which utilize mining, this is a number which is manipulated by the publishing node to solve the hash puzzle).
  - **Block data** usually contains:
    - List of transactions and ledger events included within the block.
    - Other data.

Blockchain - Components

- **Chaining Blocks**
  - Blocks are chained together through each block containing the hash digest of the previous block's header, thus forming the blockchain.



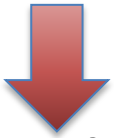
Generic Chain of Blocks

  - ***Genesis*** block – initial state of the blockchain (only pre-configured block).
  - By combining the initial state and the ability to verify every block since then, users can independently agree on the current state of the blockchain.

No need to have a trusted third party provide the state of the system.

Blockchain – Consensus Models

- Key aspect of blockchain technology is determining which user publishes the next block.



- Solved through implementing one of many possible **consensus models**, to enable a group of mutually distrusting users to work together:
  - **Permissionless blockchain networks** – consensus model to determine which participant adds the next block to the chain is resource intensive, since there are generally many publishing nodes competing at the same time to publish the next block. They usually do this to win cryptocurrency and/or transaction fees.
  - **Permissioned blockchain networks** – no need for a resource intensive consensus model to determine which participant adds the next block to the chain, since there may exist some level of trust between publishing nodes.
- Every block must be added to the blockchain after the genesis block, based on the agreed-upon consensus model of the blockchain.

Blockchain – Proof of Work Consensus Model

- In the **proof of work** (PoW) model, a user publishes the next block by being the first to solve a computationally intensive puzzle.
  - The solution to this puzzle is the "proof" they have performed work.
  - Puzzle is designed such that solving the puzzle is difficult but checking that a solution is valid is easy.
  - Enables all nodes to easily validate any proposed next blocks, and any proposed block that did not satisfy the puzzle would be rejected.
- Often the publishing nodes attempt to solve this computationally difficult puzzle to claim a reward of some sort (usually in the form of a cryptocurrency offered by the blockchain network).
- Important aspect: the work put into a puzzle does not influence one's likelihood of solving the current or future puzzles because the puzzles are independent.
- A common puzzle method is to require that the hash digest of a block header be less than a target value.
  - Publishing nodes make many small changes to their block header (e.g., changing the nonce) trying to find a hash digest that meets the requirement.
  - For each attempt, the publishing node must compute the hash for the entire block header.
  - Hashing the block header many times becomes a computationally intensive process.

Implementations: Bitcoin, Ethereum, …

Blockchain – Proof of Stake Consensus Model

- The **proof of stake** (PoS) model is based on the idea that the more stake a user has invested into the system, the more likely they will want the system to succeed, and the less likely they will want to subvert it.
  - Stake is often an amount of cryptocurrency that the blockchain network user has invested into the system
- Users with more stake are more likely to publish new blocks.
  - No need to perform resource intensive computations (involving time, electricity, and processing power) as found in proof of work.
- Reward for block publication is usually the earning of user provided transaction fees.

Implementations: Ethereum Casper, Krypton, Bitshares, …

Blockchain – Round Robin Consensus Model

- The **Round Robin** model is used by some permissioned blockchain networks.
- Nodes take turns in creating blocks.
- Ensures no one node creates the majority of the blocks.
- Lacks cryptographic puzzles, and has low power requirements.

Implementations: MultiChain

Blockchain – Proof of Authority Consensus Model

- The **Proof of Authority/Identity** model relies on the partial trust of publishing nodes through their known link to real world identities.
  - Publishing nodes must have their identities proven and verifiable within the blockchain network.
- Idea is that the publishing node is staking its identity/reputation to publish new blocks.
- Only applies to permissioned blockchain networks with high levels of trust.

Implementations: Ethereum Kovan testnet, POA Chain, Parity

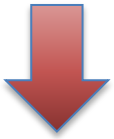
Blockchain – Proof of Elapsed Time Consensus Model

- In the **Proof of Elapsed Time** model each publishing node requests a wait time from a secure hardware time source within their computer system.
  - The secure hardware time source (usually trusted execution environment found on some computer processors, such as Intel's Software Guard Extensions, or AMD's Platform Security Processor, or ARM's TrustZone) will generate a random wait time and return it to the publishing node software.
- Publishing nodes take the random time they are given and become idle for that duration.
- Once a publishing node wakes up from the idle state, it creates and publishes a block to the blockchain network, alerting the other nodes of the new block; any publishing node that is still idle will stop waiting, and the entire process starts over.

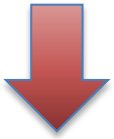
Implementations: Hyperledger, Sawtooth

Blockchain – Ledger Conflicts and Resolutions

- For some blockchain networks it is possible that multiple (different) blocks will be published at approximately the same time.
  - Permissionless blockchain networks are more prone to have conflicts due to their openness and number of competing publishing nodes.



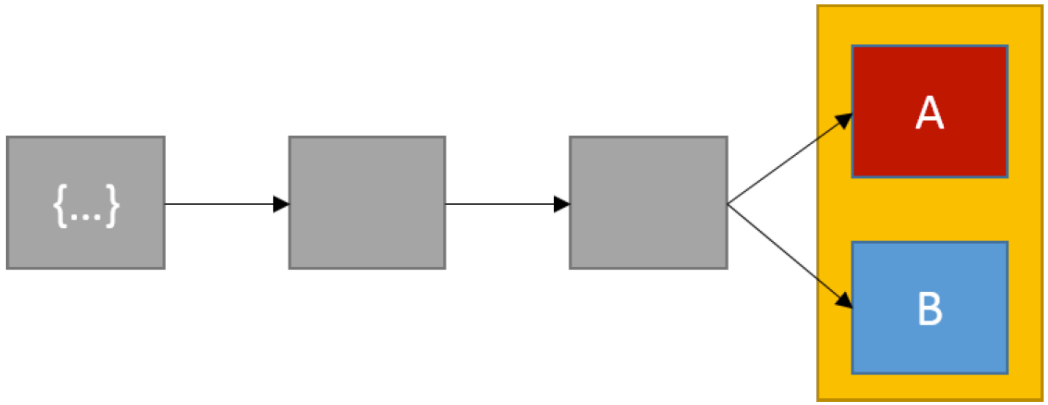
- Can cause differing versions of a blockchain to exist at any given moment.



- Must be resolved quickly to have consistency in the blockchain network.

Blockchain – Ledger Conflicts and Resolutions

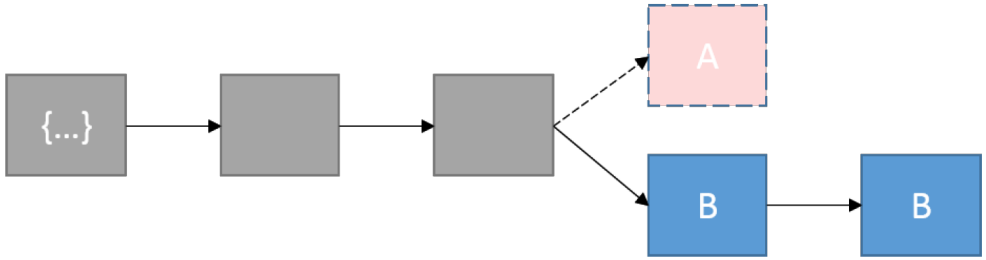
- Example:
  - *node_A* creates *block_n(A)* with transactions #1, 2 and 3. node_A distributes it to some nodes.
  - *node_B* creates *block_n(B)* with transactions #1, 2 and 4. node_B distributes it to some nodes.
  - There is a conflict (but these differing versions are not “wrong” since they were created with the information each node had available).
    - *block_n* will not be the same across the network.



Ledger in Conflict

Blockchain – Ledger Conflicts and Resolutions

- Example:
  - **Conflicts** are usually **quickly resolved**.
  - Most blockchain networks will wait until the next block is published and use that chain as the “official” blockchain, thus adopting the “longer blockchain”.



The chain with block_n(B) adds the next block, the chain with block_n(A) is now orphaned

  - Any transaction that was present in block_n(A), the orphaned block, but not present in the block_n(B) chain, is returned to the pending transaction pool (which is where all transactions which have not been included within a block reside).

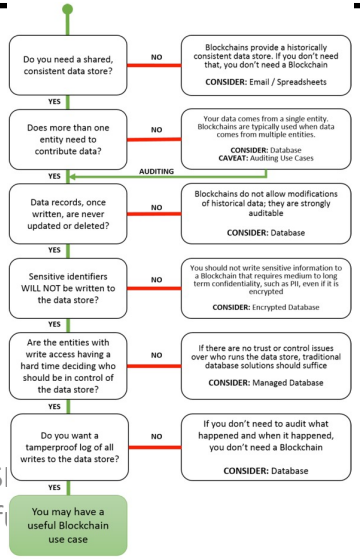
- Due to the possibility of blocks being overwritten, a transaction is not usually accepted as confirmed until several additional blocks have been created on top of the block containing the relevant transaction.
- The more blocks that have been built on top of a published block, the more likely it is that the initial block will not be overwritten.

Blockchain – Smart contracts

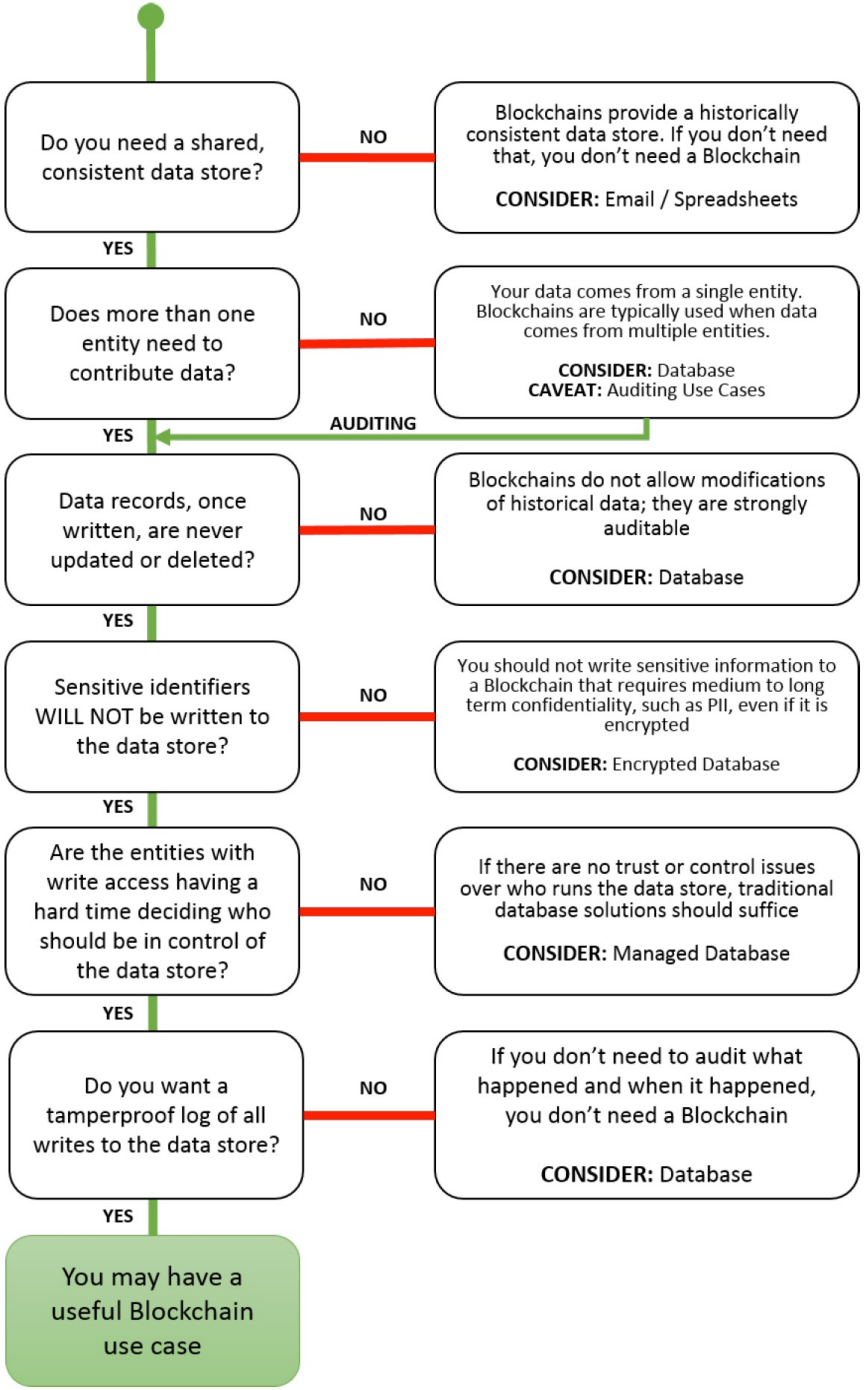
- **Smart contracts** extend and leverage blockchain technology.
- Smart contract is a collection of code and data (sometimes referred to as functions and state) that is deployed using cryptographically signed transactions on the blockchain network (e.g., Ethereum's smart contracts, Hyperledger Fabric's chaincode).
- The smart contract is executed by nodes within the blockchain network; all nodes that execute the smart contract must derive the same results from the execution, and the results of execution are recorded on the blockchain.
  - Smart contracts must be deterministic, in that given an input they will always produce the same output based on that input.
  - Additionally, all the nodes executing the smart contract must agree on the new state that is obtained after the execution. To achieve this, smart contracts cannot operate on data outside of what is directly passed into it (e.g., smart contracts cannot obtain web services data from within the smart contract – it would need to be passed in as a parameter).
- Blockchain network **users** can create **transactions** which send data to public functions offered by a smart contract.
- The smart contract executes the appropriate method with the user provided data to perform a service.
- The code, being on the blockchain, is tamper evident and tamper resistant and therefore can be used as a trusted third party.

Blockchain Technology – Where does it fit?

- Since blockchain technology is still new, a lot of organizations are looking at ways to incorporate it into their businesses.
- The fear of missing out on this technology is quite high, and most organizations approach the problem as "**we want to use blockchain somewhere, where can we do that?**" which leads to frustrations with the technology as it cannot be applied universally.
- Flowchart to help determine whether a blockchain may be needed for a development initiative:



Blockcha ...oes it fit?



DHS Science & Technology Directorate Flowchart

Blockchain

Bibliografia / Bibliography:

- NIST.IR 8202 – Blockchain Technology Overview,
  (https://nvlpubs.nist.gov/nistpubs/ir/2018/NIST.IR.8202.pdf)