



Mestrado em Engenharia Informática (MEI)

Mestrado Integrado em Engenharia Informática

(MiEI)

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

Engenharia de Segurança





Tópicos

- Criptografia Aplicada
 - Assinaturas cegas (blind signatures)
 - Protocolos/aplicações criptográficas
 - SSL/TLS
 - SSH



Assinaturas cegas (*Blind signatures*)

- Introduzidas por David Chaum, em 1982, para utilização na área da moeda e pagamentos eletrónicos.
- A assinatura cega possibilita que uma entidade peça a uma terceira entidade para assinar digitalmente uma mensagem, sem lhe revelar o conteúdo da mensagem;
- Tipicamente é efetuada a seguinte analogia com o Mundo físico:
 - É colocado um papel com a mensagem dentro de um envelope;
 - É inserido no envelope, entre a frente do envelope e o papel com a mensagem, um papel químico;
 - O envelope é fechado e fornecido ao assinante;
 - O assinante assina a frente do envelope e devolve o envelope;
 - O dono da mensagem retira o papel de dentro do envelope, e o mesmo contém a assinatura do assinante.
 - Conclusão: O assinante não viu a mensagem, mas uma terceira parte que receba a mensagem pode validar a assinatura.

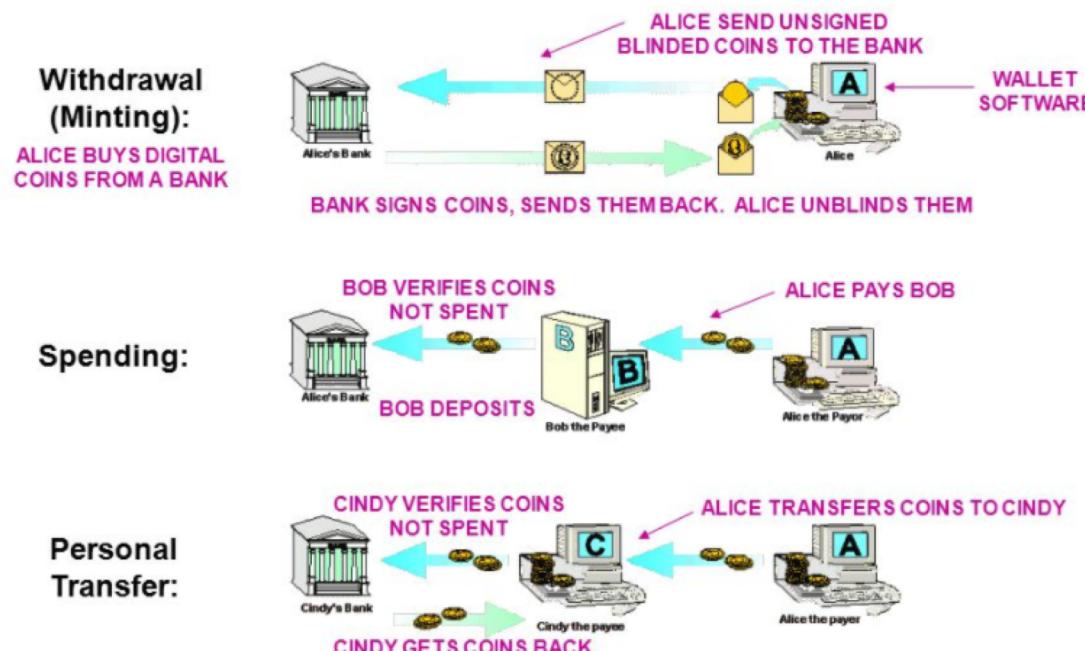


Assinaturas cegas (*Blind signatures*)

Em que situações necessito de assinaturas cegas?

- Voto eletrónico, em que os boletins de voto são assinados pelo sistema de voto antes de serem depositados na urna eletrónica, mas sem que seja revelado o conteúdo do voto.
- Em sistemas de dinheiro eletrónico, não rastreável.

eCash (Formerly DigiCash)



Assinaturas cegas (*Blind signatures*)

Como funciona?

- Suponha que Alice quer que Bob assine uma mensagem m , mas não quer que o Bob conheça o conteúdo da mensagem.
- Alice “cega/ofusca” a mensagem m , com um número aleatório b (fator de ofuscação): $\text{blind}(m, b)$
- Bob assina a mensagem com a sua chave privada d , tendo como resultado $\text{sign}(\text{blind}(m, b), d)$
- Alice desofusca a mensagem utilizando b , tendo como resultado $\text{unblind}(\text{sign}(\text{blind}(m, b), d), b)$
- As funções utilizadas estão desenhadas de tal modo que $\text{unblind}(\text{sign}(\text{blind}(m, b), d), b) = \text{sign}(m, d)$, i.e., a assinatura de m pelo Bob.
- Qualquer entidade/pessoa pode utilizar a chave pública de Bob para verificar se a assinatura é autêntica.



Assinaturas cegas (*Blind signatures*)

Assinaturas cegas RSA

- Seja e o expoente público RSA do assinante Bob, d o expoente secreto RSA do assinante Bob, e N o modulus RSA.
- Alice escolhe um valor aleatório r , tal que r é primo de N (i.e., $\gcd(r, N) = 1$).
- Alice utiliza $r^e \bmod N$ como fator de ofuscação
 - Note que como r é um valor aleatório, $r^e \bmod N$ também é aleatório.
- Seja m a mensagem que Alice quer que Bob assine, sem conhecer o seu conteúdo.



Assinaturas cegas (*Blind signatures*)

Assinaturas cegas RSA

- Alice ofusca a mensagem m com o fator de ofuscação

$$m' \equiv m r^e \pmod{N}$$
 - Note que como $r^e \pmod{N}$ é aleatório, m' não revela nenhuma informação de m
- Alice envia m' ao assinante Bob.
- O assinante utiliza o expoente secreto d para calcular a assinatura cega s'

$$s' \equiv (m')^d \pmod{N}$$
- Bob devolve s' a Alice.
- Alice remove o fator de ofuscação (utilizando r^{-1}), obtendo s , que é a assinatura de m por Bob

$$\begin{aligned}
 s' r^{-1} \pmod{N} &\equiv (m')^d r^{-1} \pmod{N} \equiv (m r^e)^d r^{-1} \pmod{N} \equiv \\
 m^d r^{ed} r^{-1} \pmod{N} &\equiv m^d r r^{-1} \pmod{N} \equiv m^d \pmod{N}
 \end{aligned}$$

Note que as chaves RSA satisfazem $r^{ed} \equiv r \pmod{N}$



Assinaturas cegas (*Blind signatures*)

Assinaturas cegas RSA

- É possível ser atacada.
- O atacante fornece ao Bob a versão cega da mensagem m' ofuscada, i.e., m'' .

$$\begin{aligned} m'' &= m'r^e \pmod{n} \\ &= (m^e \pmod{n}) \cdot r^e \pmod{n} \\ &= (mr)^e \pmod{n} \end{aligned}$$
- Quando o atacante desofusca a assinatura cega de m'' , é possível obter facilmente a mensagem inicial.

$$m = s' \cdot r^{-1} \pmod{n}, \text{ já que}$$

$$\begin{aligned} s' &= m''^d \pmod{n} \\ &= ((mr)^e \pmod{n})^d \pmod{n} \\ &= (mr)^{ed} \pmod{n} \\ &= m \cdot r \pmod{n}, \text{ since } ed \equiv 1 \pmod{\phi(n)} \end{aligned}$$

- Solução: Em vez de assinar a mensagem, assinar hash da mensagem.



Assinaturas cegas (*Blind signatures*)

Assinaturas cegas baseadas no *Elliptic Curve Discrete Logarithm Problem* (ECDLP)

- Sistema eficiente de assinatura cega, baseado na criptografia de curvas elípticas (ECC), descrito em <http://www.ijimt.org/papers/556-IT302.pdf>.
- Tem as seguintes fases:
 - Inicialização;
 - Ofuscação;
 - Assinatura;
 - Desofuscação;
 - Verificação.
- Tem três participantes:
 - Requerente (Alice),
 - Assinante (Bob),
 - Verificador.



Assinaturas cegas (*Blind signatures*)

Assinaturas cegas baseadas no ECDLP

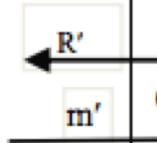
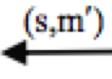
- Parâmetros da curva elíptica sobre o Corpo finito F_p são definidos da seguinte forma:

$T = (p, F_p, a, b, G, n, h)$, em que:

- p é um inteiro,
- $a, b \in F_p$ especificam a curva elíptica $E(F_p)$ definida por $y^2 = x^3 + ax + b \pmod{p}$,
- $G = (x_G, y_G)$ é um ponto base de $E(F_p)$,
- n é um número primo que define a ordem de G (i.e., número de pontos do subgrupo gerado pelo ponto base),
- h é um inteiro que define o cofator, $h = \#E(F_p)/n$ (i.e., número de pontos da curva dividido pelo número de pontos do subgrupo gerado pelo ponto base)

Assinaturas cegas (*Blind signatures*)

Assinaturas cegas baseadas no ECDLP

<i>Requester</i>	<i>Signer</i>
 Blinding Phase <p>Calculates r' from the elliptic curve point R' Integers v (randomly selected in the range $(1, n-1)$) Compute $R = v^{-1}R' = (x_0, y_0)$ $r = x_0 \bmod n$ (blinding factor) $m' = H(m) r'^{-1} r v \pmod{n}$ H is the hash function with SHA-256 algorithm</p>	 Initialization Phase <p>Publish $E_p(a, b)$ Base Point $G = (x, y)$ Integer k (randomly selected in the range $(1, n-1)$) $R' = kG = (x_1, y_1)$ and</p>
	<p>Compute $r' - x_1 \pmod{n}$ if $(r' \neq 0)$, Else if choose another k and find r'</p>
Unblinding $s' = sv^{-1}r'^{-1}r \pmod{n}$ The unblinding operation is needed to obtain the digital signature (s', R) on message m .	Signing Phase Private key = d (d randomly selected in the range $(1, n-1)$) Public key = $Q = dG = (x_Q, y_Q)$ Check (k, m') in database? If yes, re-select k . Otherwise, compute $s = dm' + kr' \pmod{n}$
	Verifying Phase Any party who has the elliptic domain parameter T of the Signer and the public key of the signer can verify the signature is genuine. $s' G^3 = QH(m) + Rr$



Assinaturas cegas (*Blind signatures*)

Assinaturas cegas baseadas no ECDLP

- Inicialização

```
user@CSI:~/Aulas/Aula3/BlindSignature$ python initSigner-app.py
Output
Init components: 7cb3aebdeaa9723f21df988a3b09fe9475f1e2253011d4cbfda09d62baa1ffe4.b22ad9e44cb42256724e82078ce59
3281bddf440d88267987af3287406bbea30
pRDashComponents: 7cb3aebdeaa9723f21df988a3b09fe9475f1e2253011d4cbfda09d62baa1ffe4.9e7f7b8503308a7565e7650f1648
b61b0b45ad2227928e770520884311d8a87d
```

- Ofuscação

```
user@CSI:~/Aulas/Aula3/BlindSignature$ python generateBlindData-app.py
Input
Data: Vamos assinar esta mensagem sem o Bob a ver
pRDash components: 7cb3aebdeaa9723f21df988a3b09fe9475f1e2253011d4cbfda09d62baa1ffe4.9e7f7b8503308a7565e7650f164
8b61b0b45ad2227928e770520884311d8a87d
Output
Blind message: 79c597b9d890706b6f48fa6279ecd5fd93ccde333257578bf74d0aed4717e374
Blind components: 2934575c9dd4f8fbbe3595e6af21bbfaa82f8aa02276acce29ee6200ca56b699.ed59ff6e354944d0c8b7d9f93fd3
f287abc56222d92a49994d600a1ccf3744ab
pRComponents: ed59ff6e354944d0c8b7d9f93fd3f287abc56222d92a49994d600a1ccf3744ab.a6157cffafe5fbbb7a41f5cda2aeb858
2568d61a1374f3c2d9cd9ddb37deb611
```



Assinaturas cegas (*Blind signatures*)

Assinaturas cegas baseadas no ECDLP

- Assinatura

```
user@CSI:~/Aulas/Aula3/BlindSignature$ python generateBlindSignature-app.py key.pem
Input
Passphrase:
Blind message: 79c597b9d890706b6f48fa6279ecd5fd93ccde333257578bf74d0aed4717e374
Init components: 7cb3aebeaa9723f21df988a3b09fe9475f1e2253011d4cbfda09d62baa1ffe4.b22ad9e44cb42256724e82078ce59
3281bddf440d88267987af3287406bbea30
Output
Blind signature: 608d64e75bd4b08733bbc03ed60ad977d063d21943de5875cce27d0d80e42a667a03f7cb347699164cf91e75b13bb
a47e226f0bd5fdbb3e762686df0cabe104
```

- Desofuscação

```
user@CSI:~/Aulas/Aula3/BlindSignature$ python unblindSignature-app.py
Input
Blind signature: 608d64e75bd4b08733bbc03ed60ad977d063d21943de5875cce27d0d80e42a667a03f7cb347699164cf91e75b13bb
a47e226f0bd5fdbb3e762686df0cabe104
Blind components: 2934575c9dd4f8fbcb3595e6af21bbfaa82f8aa02276acce29ee6200ca56b699.ed59ff6e354944d0c8b7d9f93fd3
f287abc56222d92a49994d600a1ccf3744ab
pRDash components: 7cb3aebeaa9723f21df988a3b09fe9475f1e2253011d4cbfda09d62baa1ffe4.9e7f7b8503308a7565e7650f164
8b61b0b45ad2227928e770520884311d8a87d
Output
Signature: f5efcba14b35e3f6c04a19772fa6af3f95550733ce15e84c687f14adc82b927d
```

Assinaturas cegas (*Blind signatures*)

Assinaturas cegas baseadas no ECDLP

- Verificação

```
user@CSI:~/Aulas/Aula3/BlindSignature$ python verifySignature-app.py key.crt
Input
Original data: Vamos assinar esta mensagem sem o Bob a ver
Signature: f5efcba14b35e3f6c04a19772fa6af3f95550733ce15e84c687f14adc82b927d
Blind components: 2934575c9dd4f8fbcc3595e6af21bbfaa82f8aa02276acce29ee6200ca56b699.ed59ff6e354944d0c8b7d9f93fd3
f287abc56222d92a49994d600a1ccf3744ab
pR components: ed59ff6e354944d0c8b7d9f93fd3f287abc56222d92a49994d600a1ccf3744ab.a6157cffafe5fbbb7a41f5cda2aeb85
82568d61a1374f3c2d9cd9ddb37deb611
Output
Valid signature
```

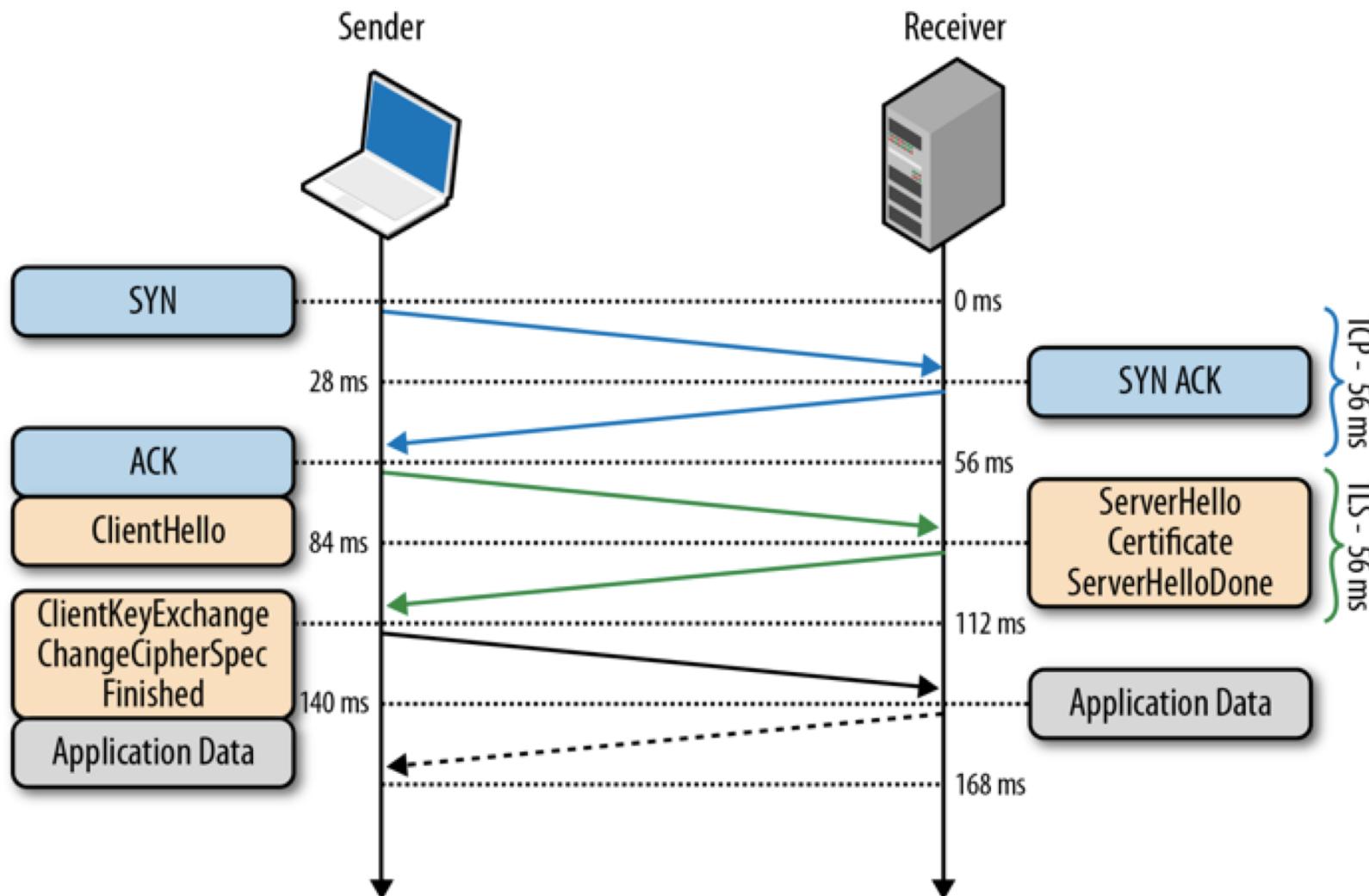




Protocolos/aplicações criptográficas

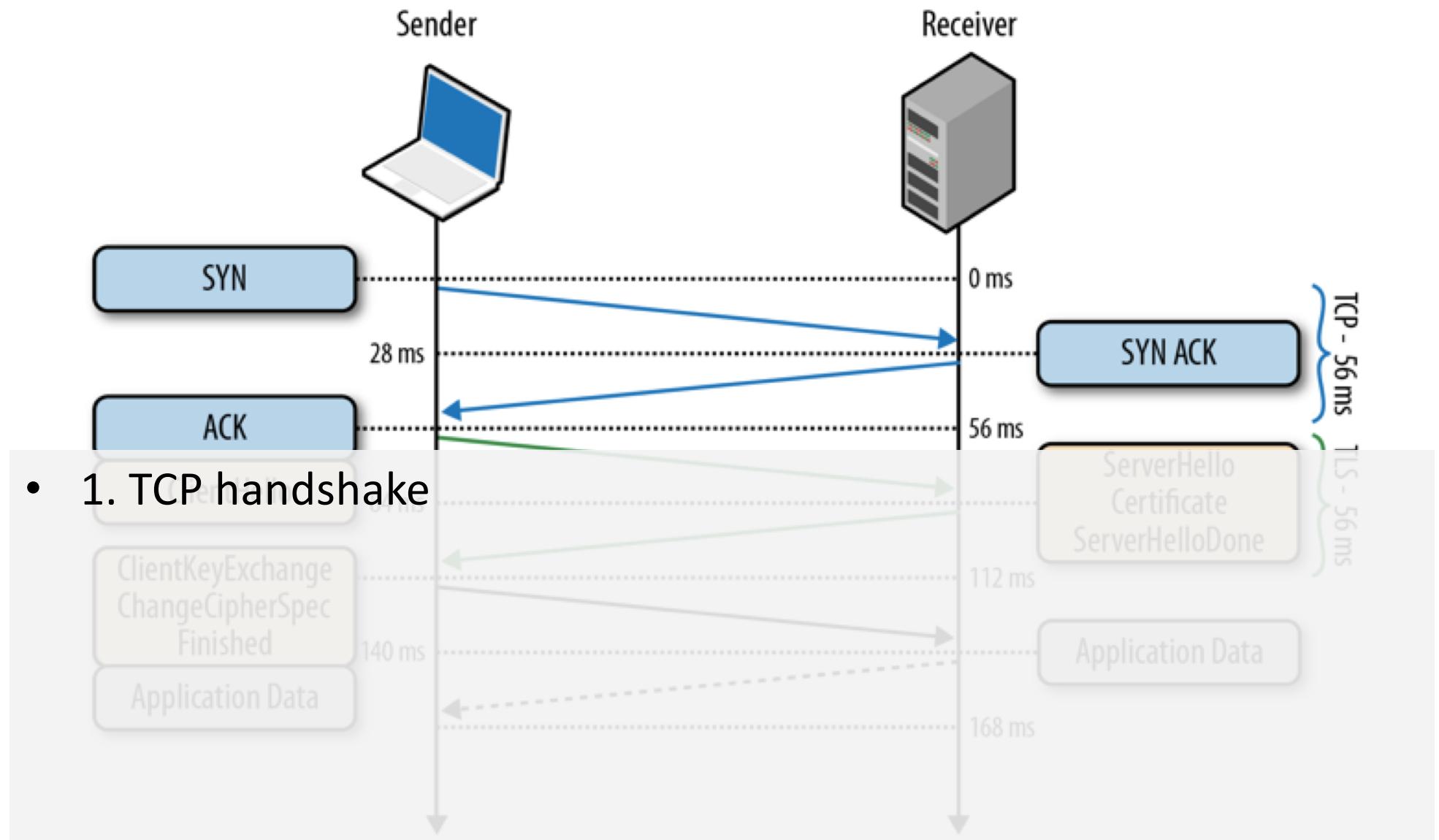


SSL/TLS (RFC 5246)



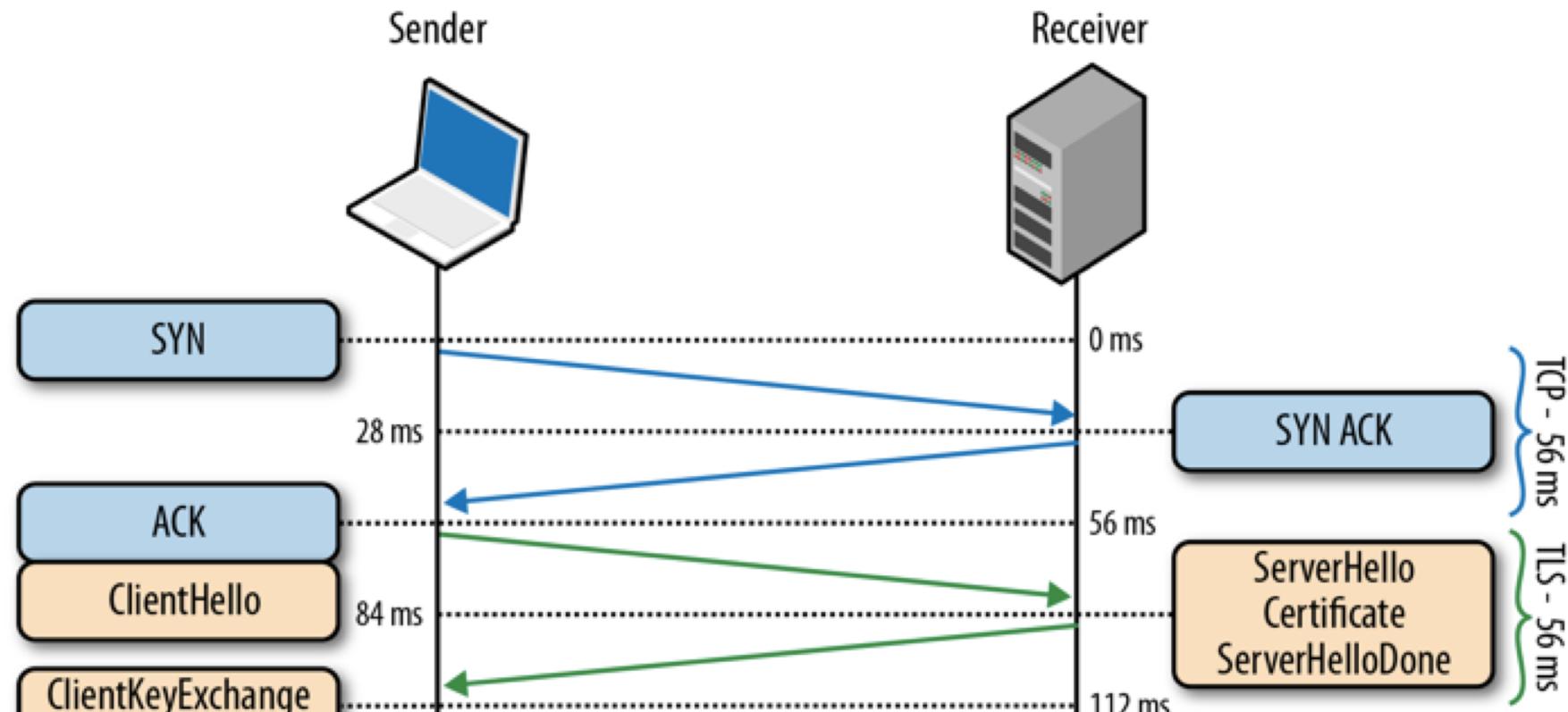
- Nota: supondo um roundtrip de 56 ms

SSL/TLS (RFC 5246)



- **1. TCP handshake**

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.

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)

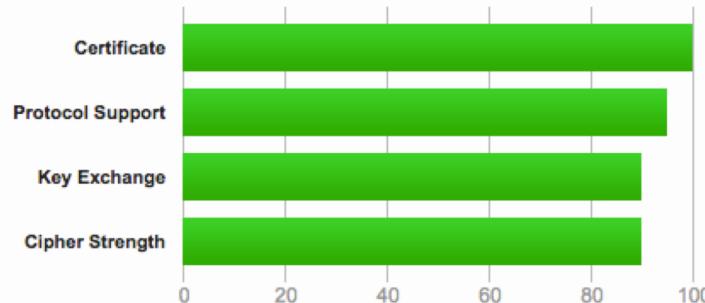
SSL Report: evotum.uminho.pt (193.137.9.162)

Assessed on: Wed, 14 Feb 2018 14:18:52 UTC | [Hide](#) | [Clear cache](#)

[Scan Another](#)

Summary

Overall Rating



Visit our [documentation page](#) for more information, configuration guides, and books. Known issues are documented [here](#).

This site works only in browsers with SNI support.

HTTP Strict Transport Security (HSTS) with long duration deployed on this server. [MORE INFO](#)

Certificate #1: RSA 2048 bits (SHA256withRSA)



Server Key and Certificate #1



evotum.uminho.pt

Fingerprint SHA256: b86922127e7a060e6a9f1cd2bbcea530031a0e12de0cfacff4ea8ce0014f501

Pin SHA256: epv64qjV7DzloS2a6Nl6rKrX3/wPHAngJAoD6LThU=





SSL/TLS – teste (www.ssllabs.com)



Protocols

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

For TLS 1.3 tests, we currently support draft version 18.



Cipher Suites

# TLS 1.2 (suites in server-preferred order)	[-]
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xc030) ECDH secp256r1 (eq. 3072 bits RSA) FS	256
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384 (0xc028) ECDH secp256r1 (eq. 3072 bits RSA) FS	256
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (0xc014) ECDH secp256r1 (eq. 3072 bits RSA) FS	256
TLS_DHE_RSA_WITH_AES_256_GCM_SHA384 (0x9f) DH 4096 bits FS	256
TLS_DHE_RSA_WITH_AES_256_CBC_SHA256 (0x6b) DH 4096 bits FS	256
TLS_DHE_RSA_WITH_AES_256_CBC_SHA (0x39) DH 4096 bits FS	256
TLS_DHE_RSA_WITH_CAMELLIA_256_CBC_SHA (0x88) DH 4096 bits FS	256





SSL/TLS – teste (www.ssllabs.com)



Handshake Simulation

Android 2.3.7	No SNI ²	Incorrect certificate because this client doesn't support SNI			
		RSA 2048 (SHA256)	TLS 1.0	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA	DH 4096
Android 4.0.4		RSA 2048 (SHA256)	TLS 1.0	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA	ECDH secp256r1 FS
Android 4.1.1		RSA 2048 (SHA256)	TLS 1.0	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA	ECDH secp256r1 FS
Android 4.2.2		RSA 2048 (SHA256)	TLS 1.0	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA	ECDH secp256r1 FS
Android 4.3		RSA 2048 (SHA256)	TLS 1.0	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA	ECDH secp256r1 FS
Android 4.4.2		RSA 2048 (SHA256)	TLS 1.2	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	ECDH secp256r1 FS
Android 5.0.0		RSA 2048 (SHA256)	TLS 1.2	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA	ECDH secp256r1 FS
Android 6.0		RSA 2048 (SHA256)	TLS 1.2 > http/1.1	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA	ECDH secp256r1 FS
Android 7.0		RSA 2048 (SHA256)	TLS 1.2	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	ECDH secp256r1 FS
Chrome 49 / XP SP3		RSA 2048 (SHA256)	TLS 1.2 > http/1.1	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA	ECDH secp256r1 FS
Chrome 57 / Win 7 R		RSA 2048 (SHA256)	TLS 1.2	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	ECDH secp256r1 FS
Firefox 31.3.0 ESR / Win 7		RSA 2048 (SHA256)	TLS 1.2	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA	ECDH secp256r1 FS
Firefox 47 / Win 7 R		RSA 2048 (SHA256)	TLS 1.2 > http/1.1	TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA	ECDH secp256r1 FS
Firefox 49 / XP SP3		RSA 2048 (SHA256)	TLS 1.2 > http/1.1	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	ECDH secp256r1 FS
Firefox 53 / Win 7 R		RSA 2048 (SHA256)	TLS 1.2	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	ECDH secp256r1 FS
Java 6u45	No SNI ²	Client does not support DH parameters > 1024 bits			
		RSA 2048 (SHA256)	TLS 1.0 TLS_DHE_RSA_WITH_AES_128_CBC_SHA	DH 4096	
Java 7u25		RSA 2048 (SHA256)	TLS 1.0	TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA	ECDH secp256r1 FS
Java 8u31		RSA 2048 (SHA256)	TLS 1.2	TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	ECDH secp256r1 FS
OpenSSL 0.9.8y		RSA 2048 (SHA256)	TLS 1.0	TLS_DHE_RSA_WITH_AES_256_CBC_SHA	DH 4096 FS
OpenSSL 1.0.1l R		RSA 2048 (SHA256)	TLS 1.2	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	ECDH secp256r1 FS
OpenSSL 1.0.2e R		RSA 2048 (SHA256)	TLS 1.2	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	ECDH secp256r1 FS
Safari 9 / iOS 9 R		RSA 2048 (SHA256)	TLS 1.2 > http/1.1	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	ECDH secp256r1 FS
Safari 9 / OS X 10.11 R		RSA 2048 (SHA256)	TLS 1.2 > http/1.1	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	ECDH secp256r1 FS
Safari 10 / iOS 10 R		RSA 2048 (SHA256)	TLS 1.2 > http/1.1	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	ECDH secp256r1 FS
Safari 10 / OS X 10.12 R		RSA 2048 (SHA256)	TLS 1.2 > http/1.1	TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384	ECDH secp256r1 FS





SSL/TLS – teste (openssl)

```
user@CSI:~$ openssl s_client -state -servername evotum.uminho.pt -connect evotum.uminho.pt:443
CONNECTED(00000003)
SSL_connect:before SSL initialization
SSL_connect:SSLv3/TLS write client hello
SSL_connect:SSLv3/TLS write client hello
SSL_connect:SSLv3/TLS read server hello
```

```
---
Certificate chain
0 s:/C=PT/ST=Braga/L=Braga/O=Universidade do Minho/OU=SCOM/CN=evotum.uminho.pt
  i:/C=NL/ST=Noord-Holland/L=Amsterdam/O=TERENA/CN=TERENA SSL CA 3
1 s:/C=US/O=DigiCert Inc/OU=www.digicert.com/CN=DigiCert Assured ID Root CA
  i:/C=US/O=DigiCert Inc/OU=www.digicert.com/CN=DigiCert Assured ID Root CA
2 s:/C=NL/ST=Noord-Holland/L=Amsterdam/O=TERENA/CN=TERENA SSL CA 3
  i:/C=US/O=DigiCert Inc/OU=www.digicert.com/CN=DigiCert Assured ID Root CA
---
```





SSL/TLS – teste (openssl)

```
user@CSI:~$ openssl s_client -state -servername evotum.uminho.pt -connect evotum.uminho.pt:443
CONNECTED(00000003)
SSL_connect:before SSL initialization
SSL_connect:SSLv3/TLS write client hello
SSL_connect:SSLv3/TLS write client hello
SSL_connect:SSLv3/TLS read server ---
                                SSL handshake has read 4262 bytes and written 327 bytes
                                Verification: OK
---
Certificate chain
 0 s:/C=PT/ST=Braga/L=Braga/O=UnivServer public key is 2048 bit
    i:/C=NL/ST=Noord-Holland/L=AmstSecure Renegotiation IS supported
 1 s:/C=US/O=DigiCert Inc/OU=www.dCompression: NONE
    i:/C=US/O=DigiCert Inc/OU=www.dExpansion: NONE
 2 s:/C=NL/ST=Noord-Holland/L=AmstNo ALPN negotiated
    i:/C=US/O=DigiCert Inc/OU=www.dSSL-Session:
      Protocol : TLSv1.2
      Cipher   : ECDHE-RSA-AES256-GCM-SHA384
      Session-ID: 17595A5243D0ED2AEA2345C32E5E6392B9B20C0AA664906F7C784825F9C4F818
      Session-ID-ctx:
      Master-Key: 81905ADC8F48E67A0EFA2B3F87B1C7F1D8B3D4517087F7DED26425B4B10EB0479FDBA0ED57E8B03AD59
      DACCC68DB8102F
      PSK identity: None
      PSK identity hint: None
      SRP username: None
      TLS session ticket lifetime hint: 300 (seconds)
      TLS session ticket:
      0000 - 2d 4a e6 31 8b 37 d3 f4-2e 68 c2 ad 70 0e 5b e2 -J.1.7...h..p.[.
      0010 - 89 41 0f 70 7f 13 2f d1-9e 57 b8 0c a7 4c f3 bf .A.p.../..W...L..
      0020 - 2b 9a 51 48 a9 b4 3e 4e-1f 53 46 9d b2 ce 13 7f +.QH..>N.SF.....
      0030 - 23 4e af a1 bc 4b 5b 0c-79 82 5d 3f 48 66 de 65 #N...K[.y.]?Hf.e
      0040 - 6c e0 44 2c a9 2d 85 58-43 eb 37 30 7a fa 7f cc l.D,..-.XC.70z...
      0050 - f5 4e 06 ab a6 44 b4 52-39 fb 94 3f 36 50 f0 0c .N...D.R9..?6P..
      0060 - 80 32 54 8a 03 f3 51 5a-62 4e 27 f1 e7 fe ef df .2T...QZbN'.....
      0070 - de 36 bb 5e 8b b8 d6 3a-96 95 25 bb 37 e5 ba bc .6.^....%..7...
      0080 - 54 42 3d 29 4d fd c1 e0-9c 06 ca 2a 0c 50 8a a2 TB=)M.....*.P..
      0090 - 27 b7 0e db 43 12 a6 8e-8f 63 6a 16 32 d2 18 d1 '...C....cj.2...
      00a0 - 85 d0 7a 17 4a d9 aa 05-3b 3b 07 85 91 ab fc f8 ..z.J....;;
      00b0 - 16 b6 2b b4 18 ae e5 7c-52 25 23 3f 7d ca 19 c4 ..+....|R%#?}...

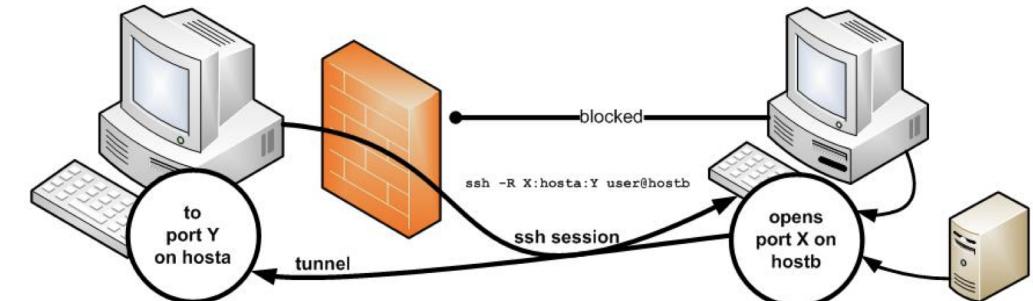
      Start Time: 1518960431
      Timeout   : 7200 (sec)
      Verify return code: 0 (ok)
      Extended master secret: no
```



2020, ---

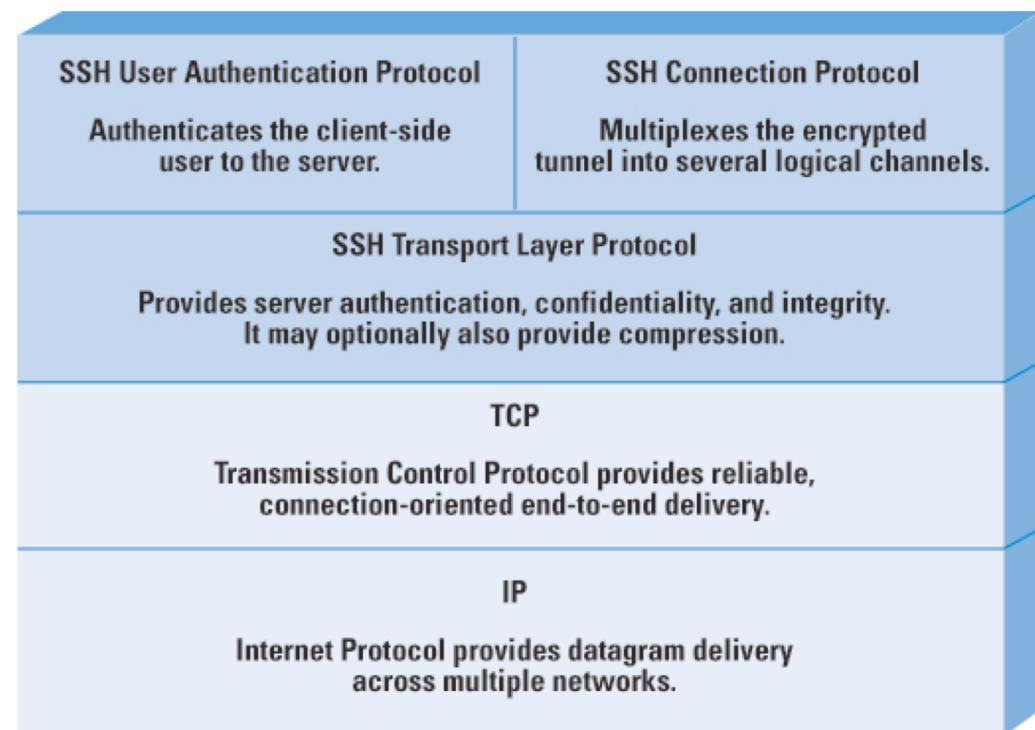
jose.miranda@devisefutures.com

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)

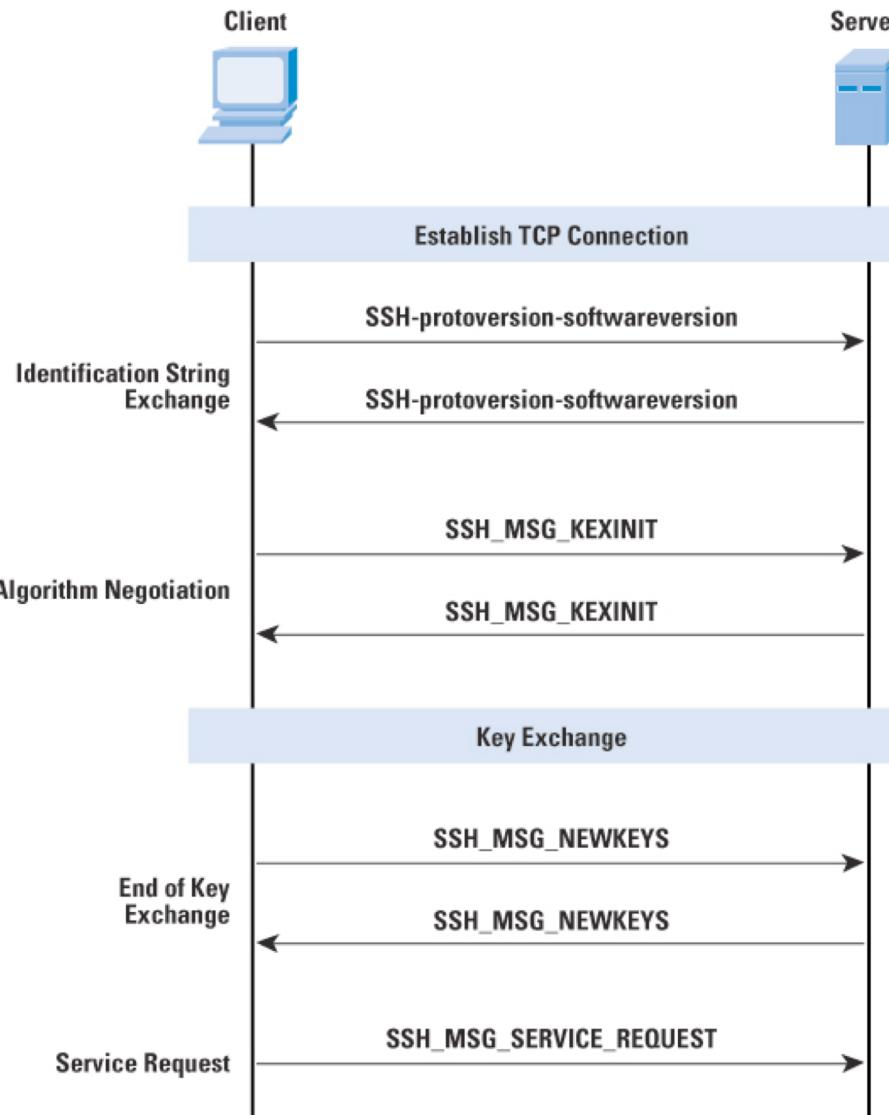
- Organizado em três protocolos sobre o TCP
 - SSH Transport Layer Protocol
 - Nota: tem a propriedade de *forward secrecy* (i.e., se uma chave for comprometida durante uma sessão, esse conhecimento não afecta a segurança das sessões anteriores)
 - SSH User Authentication Protocol
 - SSH Connection Protocol



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.

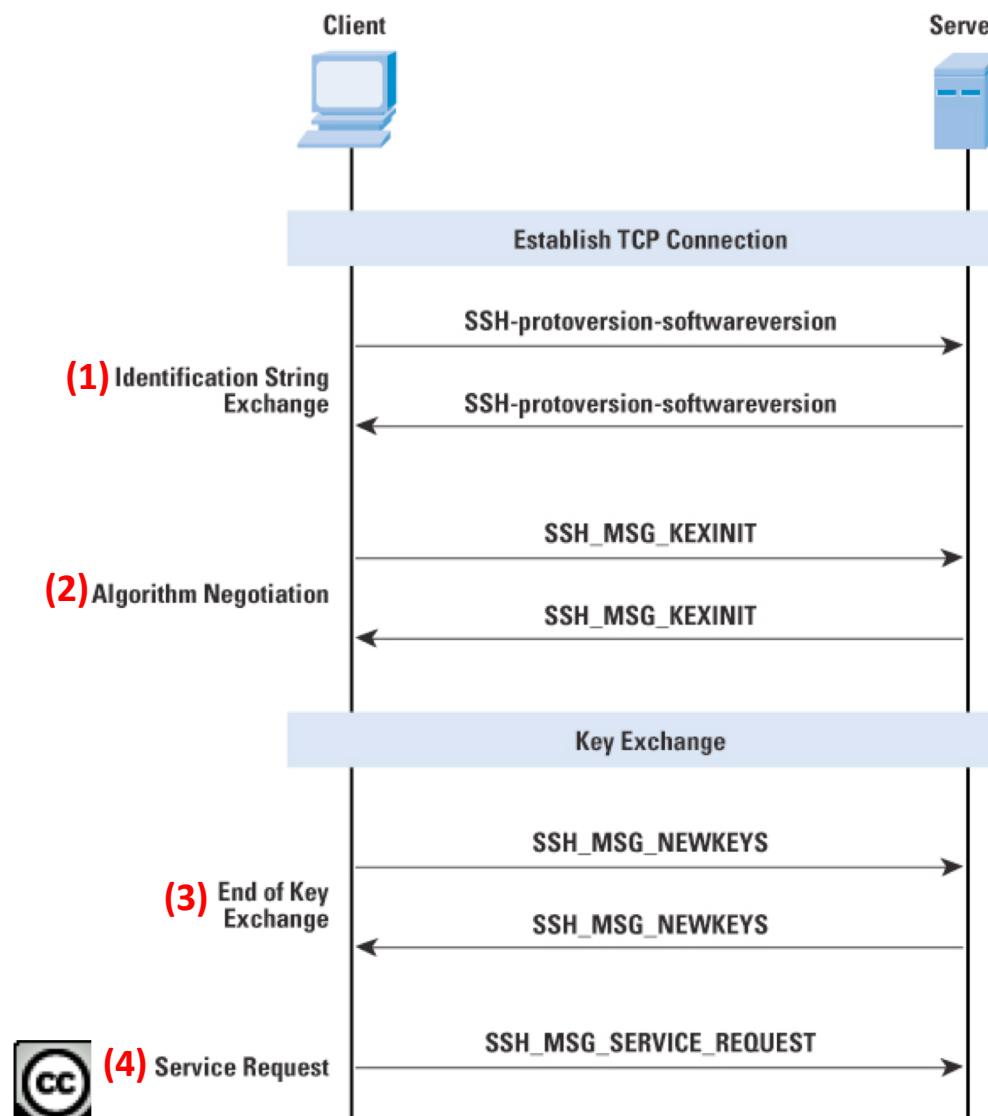


- A autenticação do servidor é efectuada baseada no seu par de chaves pública-privada;
- Para esta autenticação ser possível, o cliente tem que conhecer (previamente) a chave pública do servidor
 - O mais normal é o cliente ter uma base de dados local com as chaves públicas dos servidores conhecidos, 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

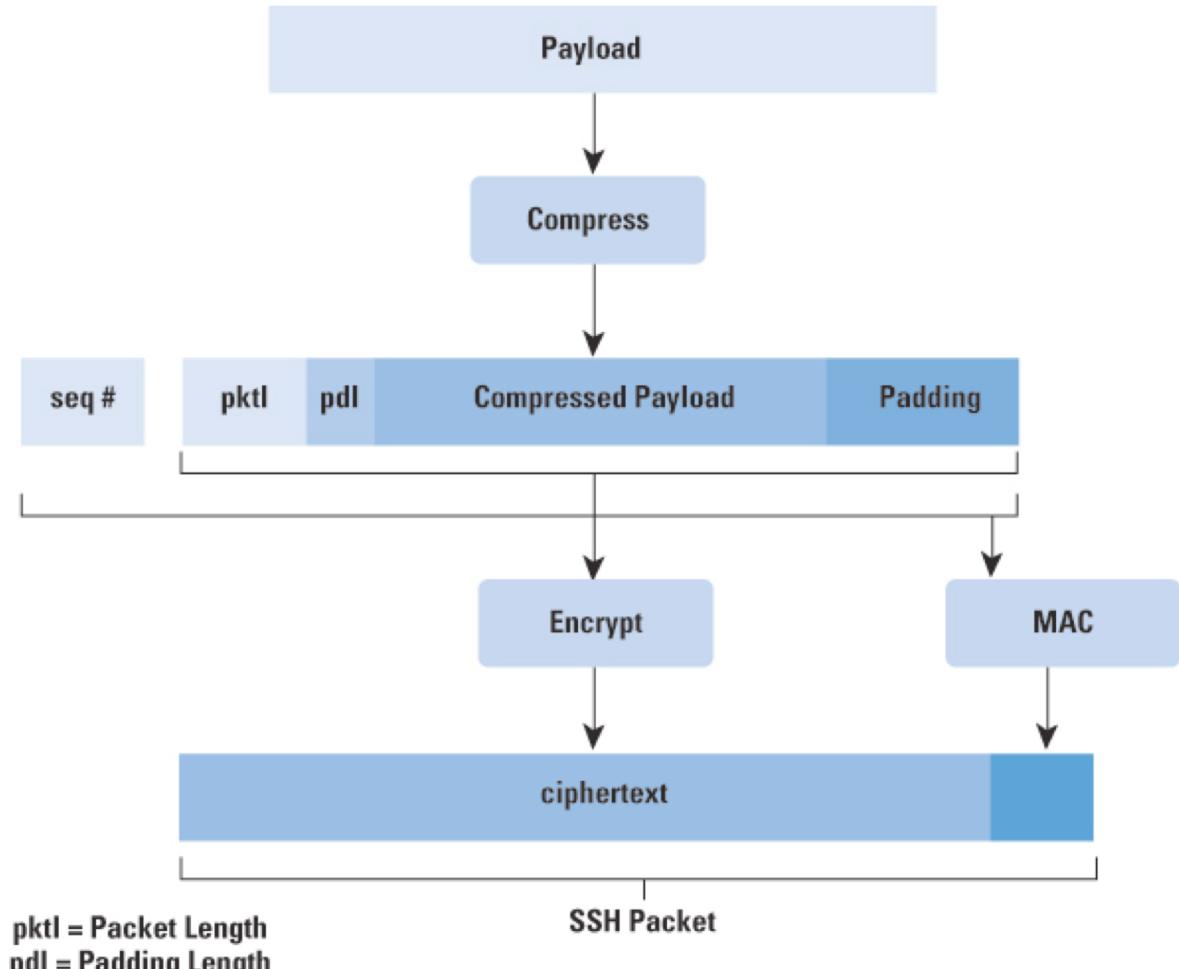
Packet length: tamanho do pacote SSH em bytes, sem incluindo os campos *packet length* e MAC.

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](#))

SSH User Authentication Protocol
Authenticates the client-side user to the server.

- 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.

byte	<code>SSH_MSG_USERAUTH_REQUEST (50)</code>
string	<code>username</code>
string	<code>service name</code>
string	<code>method name</code>
....	<code>method-specific fields</code>

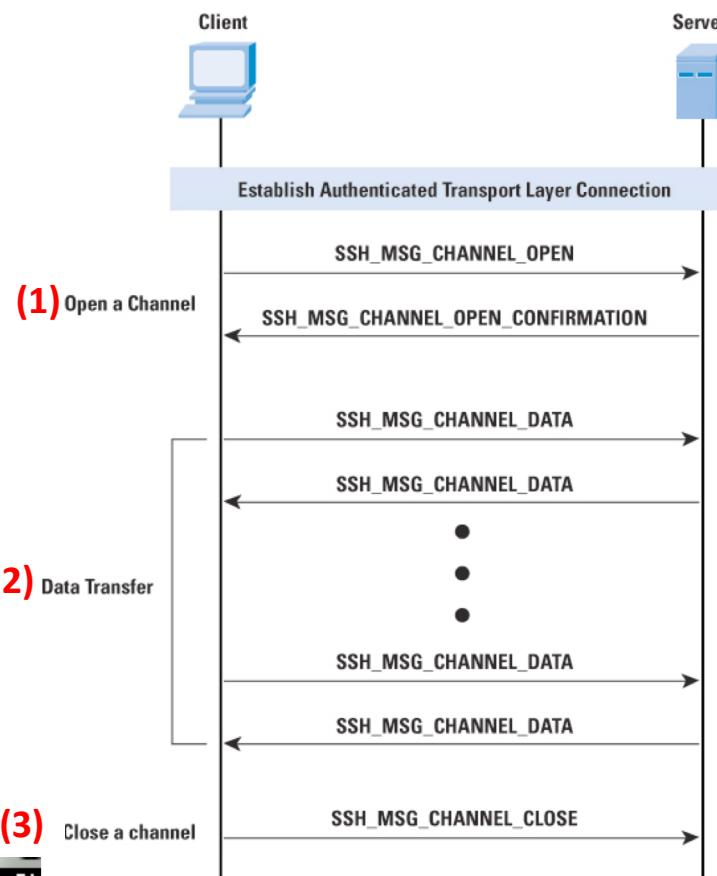


SSH (Secure Shell)

- 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:hHqIkUw3XmEUnVKmFqP4ajGeFtw0P6QJns0wEN2DZXE.
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:hHqIkUw3XmEUnVKmFqP4ajGeFtw0P6QJns0wEN2DZXE root@DF.server01 (ECDSA)
```



SSH (Secure Shell)

- Utilização:
 - Acesso remoto com autenticação do utilizador
 - 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:

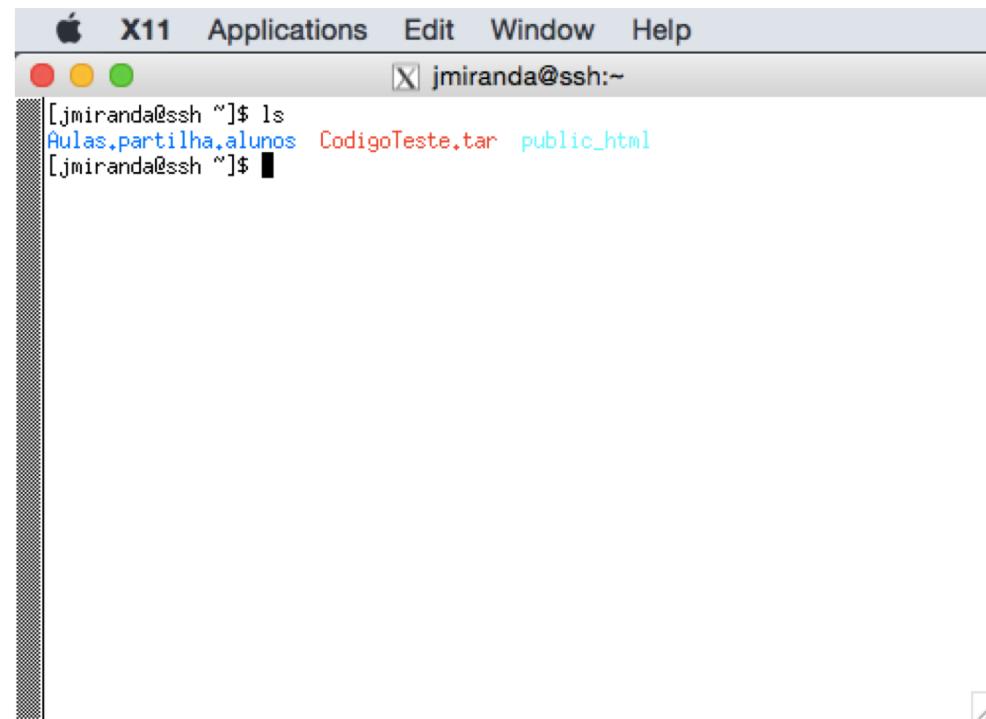
```
[jmiranda@ssh .ssh]$ ls -la
total 8
drwx----- 2 jmiranda dcc 28 Nov 26 20:29 .
drwx----- 5 jmiranda dcc 4096 Dec 15 17:19 ..
-rw-r--r-- 1 jmiranda dcc 393 Nov 26 20:29 authorized_keys
```

SSH (Secure Shell)

- Utilização:
 - Reencaminhamento de sessões X11

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

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





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*)
 - 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 remota efectuar:

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

- Na máquina local (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

2020, UMinho, EEng
jose.

```
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)

```
user@CSI:~/Tools/ssh-audit$ python ssh-audit.py ssh.dcc.fc.up.pt
# general
(gen) banner: SSH-2.0-OpenSSH_6.6.1
(gen) software: OpenSSH 6.6.1
(gen) compatibility: OpenSSH 6.5-6.6, Dropbear SSH 2013.62+ (some functionality from 0.52)
(gen) compression: enabled (zlib@openssh.com)

# key exchange algorithms
(kex) curve25519-sha256@libssh.org          -- [info] available since OpenSSH 6.5, Dropbear SSH 2013.62
(kex) ecdh-sha2-nistp256                      -- [fail] using weak elliptic curves
                                                `-- [info] available since OpenSSH 5.7, Dropbear SSH 2013.62
(kex) ecdh-sha2-nistp384                      -- [fail] using weak elliptic curves
                                                `-- [info] available since OpenSSH 5.7, Dropbear SSH 2013.62
...
(kex) ...

# host-key algorithms
(key) ssh-rsa                                -- [info] available since OpenSSH 2.5.0, Dropbear SSH 0.28
(key) ecdsa-sha2-nistp256                      -- [fail] using weak elliptic curves
                                                `-- [warn] using weak random number generator could reveal the key
                                                `-- [info] available since OpenSSH 5.7, Dropbear SSH 2013.62

# 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-etm@openssh.com               -- [fail] removed (in server) since OpenSSH 6.7, unsafe algorithm
                                                `-- [warn] disabled (in client) since OpenSSH 7.2, legacy algorithm
                                                `-- [warn] using weak hashing algorithm
                                                `-- [info] available since OpenSSH 6.2
(mac) hmac-sha1-etm@openssl.com              -- [warn] using weak hashing algorithm

# algorithm recommendations (for OpenSSH 6.6.1)
(rec) -diffie-hellman-group14-sha1           -- kex algorithm to remove
(rec) -diffie-hellman-group-exchange-sha1     -- kex algorithm to remove
(rec) -diffie-hellman-group1-sha1             -- kex algorithm to remove
(rec) -ecdh-sha2-nistp256                     -- kex algorithm to remove
(rec) -ecdh-sha2-nistp521                     -- kex algorithm to remove
(rec) -ecdh-sha2-nistp384                     -- kex algorithm to remove
(rec) -ecdsa-sha2-nistp256                    -- key algorithm to remove
(rec) +ssh-ed25519                           -- key algorithm to append
```

