Segurança de Sessão

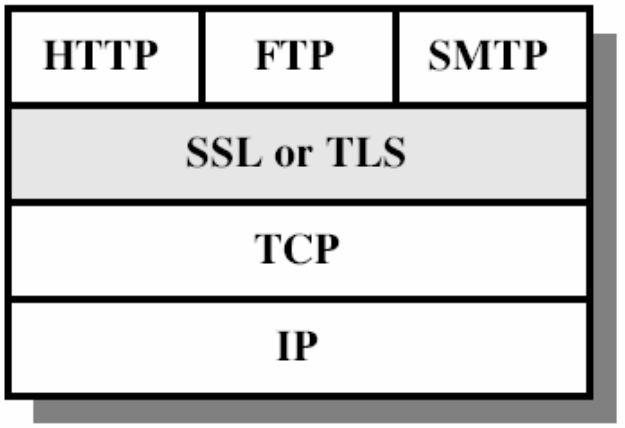
Jaime Dias

FEUP > DEEC > MRSC > **Segurança em Sistemas e Redes**

v3.1

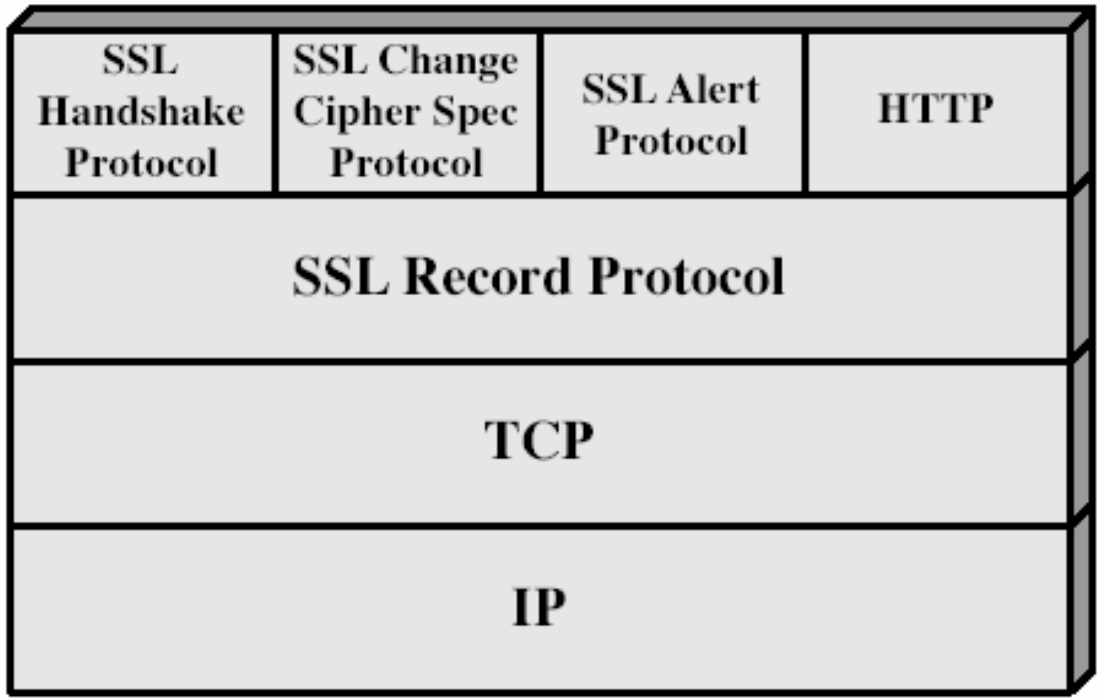
SSL/TLS

Introdução

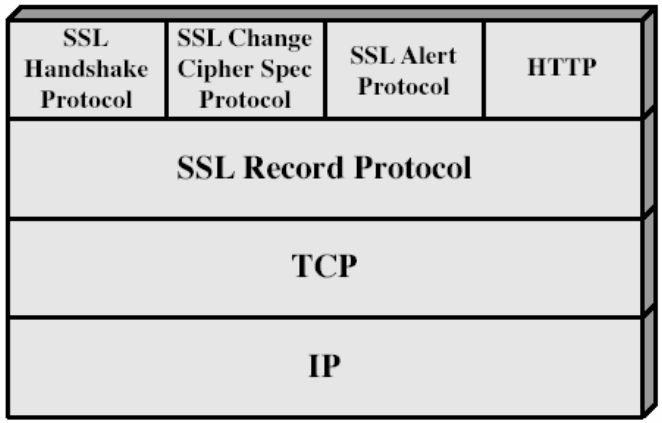


- Netscape desenvolveu SSL (*Secure Socket Layer*)
  - Versões 2 e 3
- IETF ➔ TLS 1.0 (*Transport Layer Security*)
  - TLS 1.0 corresponde à versão 3.1 do SSL
- TLS 1.0 não é compatível com SSL 3.0
  - Aplicações implementam TLS e SSL
- SSL/TLS transparente para os protocolos de aplicação
- Tipicamente ➔ protocolo sobre SSL/TLS acrescenta-se um "S".
  - ex: HTTPS, IMAPS, POP3S
- Aplicações à escuta em portos diferentes. Ex:
  - HTTP 80 → HTTPS 443
  - SMTP 25 → SMTPS 465 (25)
  - POP3 110 → POP3S 995
  - …

Pilha protocolar

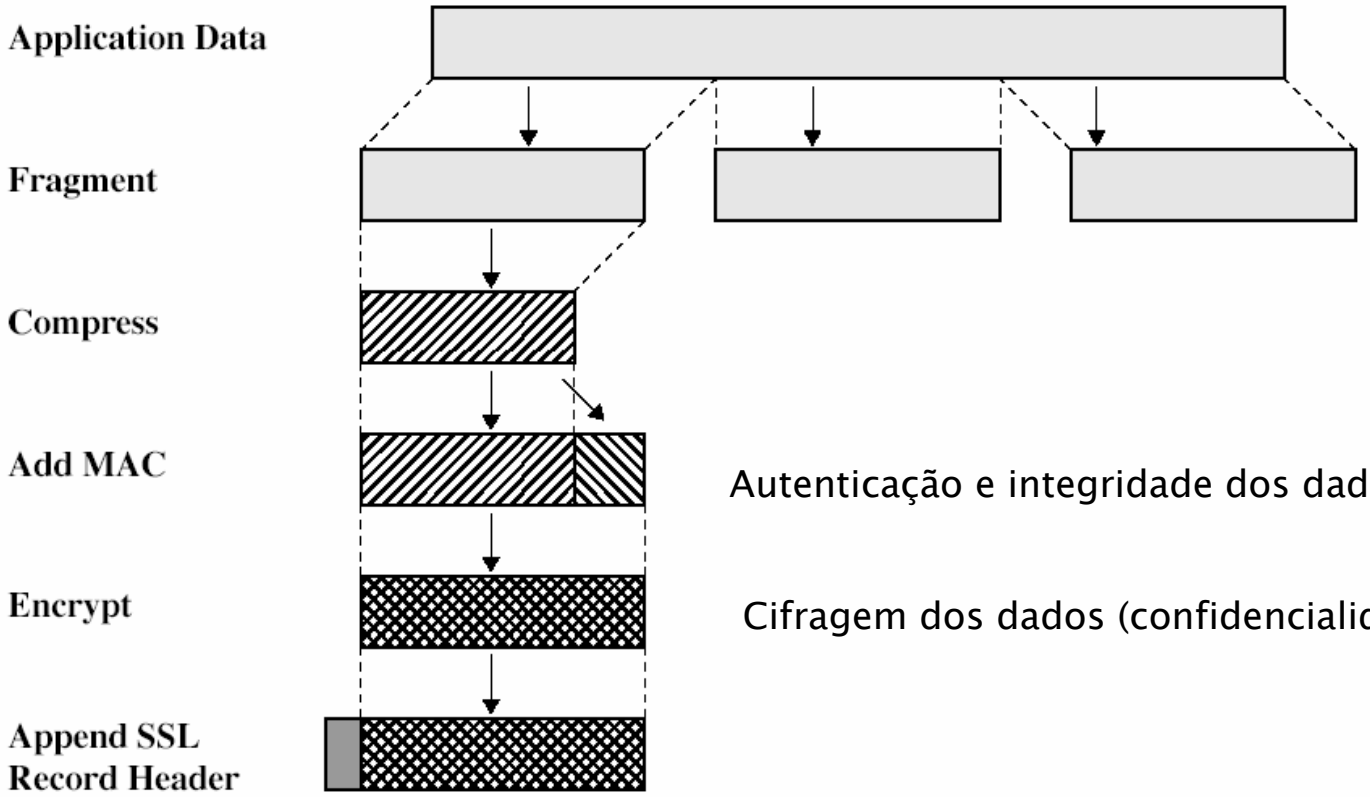


Pilha protocolar



- *Record Protocol*
  - Confidencialidade, autenticidade e integridade das mensagens
  - Fragmentação e eventual compressão das mensagens
- *Handshake Protocol*
  - Permite autenticação do cliente e servidor
  - Negociação de algoritmos criptográficos a usar em cada conexão (C*ipherSuite*)
  - Troca de chaves simétricas para confidencialidade, autenticidade e integridade (MAC)
- *Alert Protocol*
  - Tipos de alerta: *warning* ou *fatal*
  - Descrição do alerta: ex: *unexpected_message*, *bad_record_mac*
- *Change Cipher Spec protocol*
  - O protocolo mais simples ➔ 1 byte com valor 1
  - Quando recebido aplica o *CipherSuite* negociado à currente conexão

Record Protocol



Autenticação e integridade dos dados

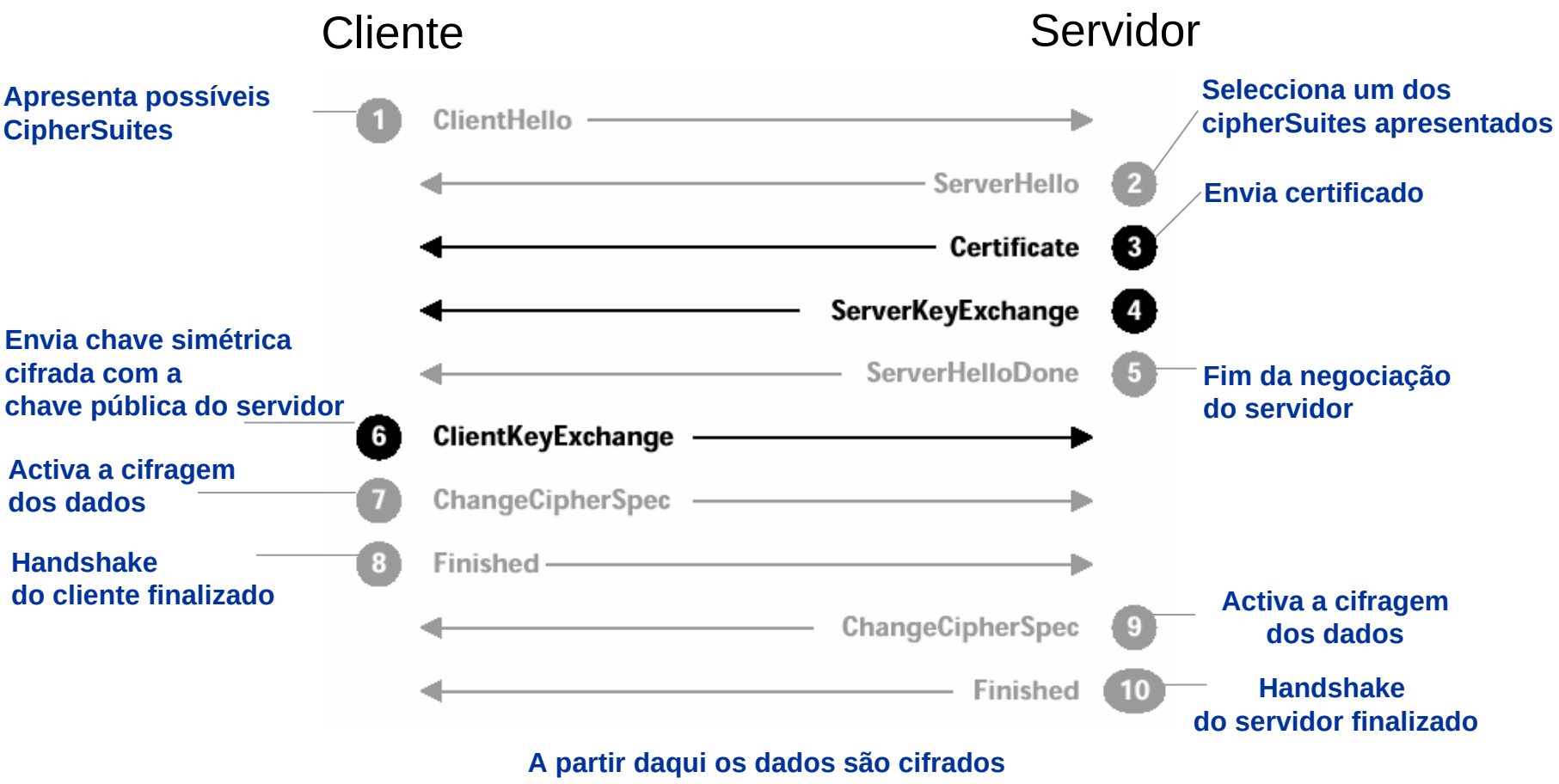
Cifragem dos dados (confidencialidade)

Handshake Protocol

3 fases

- Mensagens *Hello*
- Mensagens de troca de certificados e de geração de chaves
- Mensagens *ChangeCipherSpec* e *Finished*

Handshake Protocol



Mensagem ClientHello

- Versão do protocolo
  - SSLv3(major=3, minor=0)
  - TLS (major=3, minor=1)
- Número aleatório
  - 32 bytes
  - Primeiros 4 bytes consistem na hora do dia em segundos, os restantes 28 bytes são aleatórios
  - Previne ataques de repetição
- ID da sessão
  - 32 bytes – indica a utilização de chaves criptográficas de uma conexão anterior (mesma sessão)
- Algoritmo de compressão
  - Normalmente é nulo

Mensagem ClientHello

- CipherSuite
  - Conjunto do algoritmos a utilizar
  - O SSL/TLS suporta vários *ciphersuites*
  - O cliente propõe um ou mais *ciphersuites;* o servidor escolhe um.

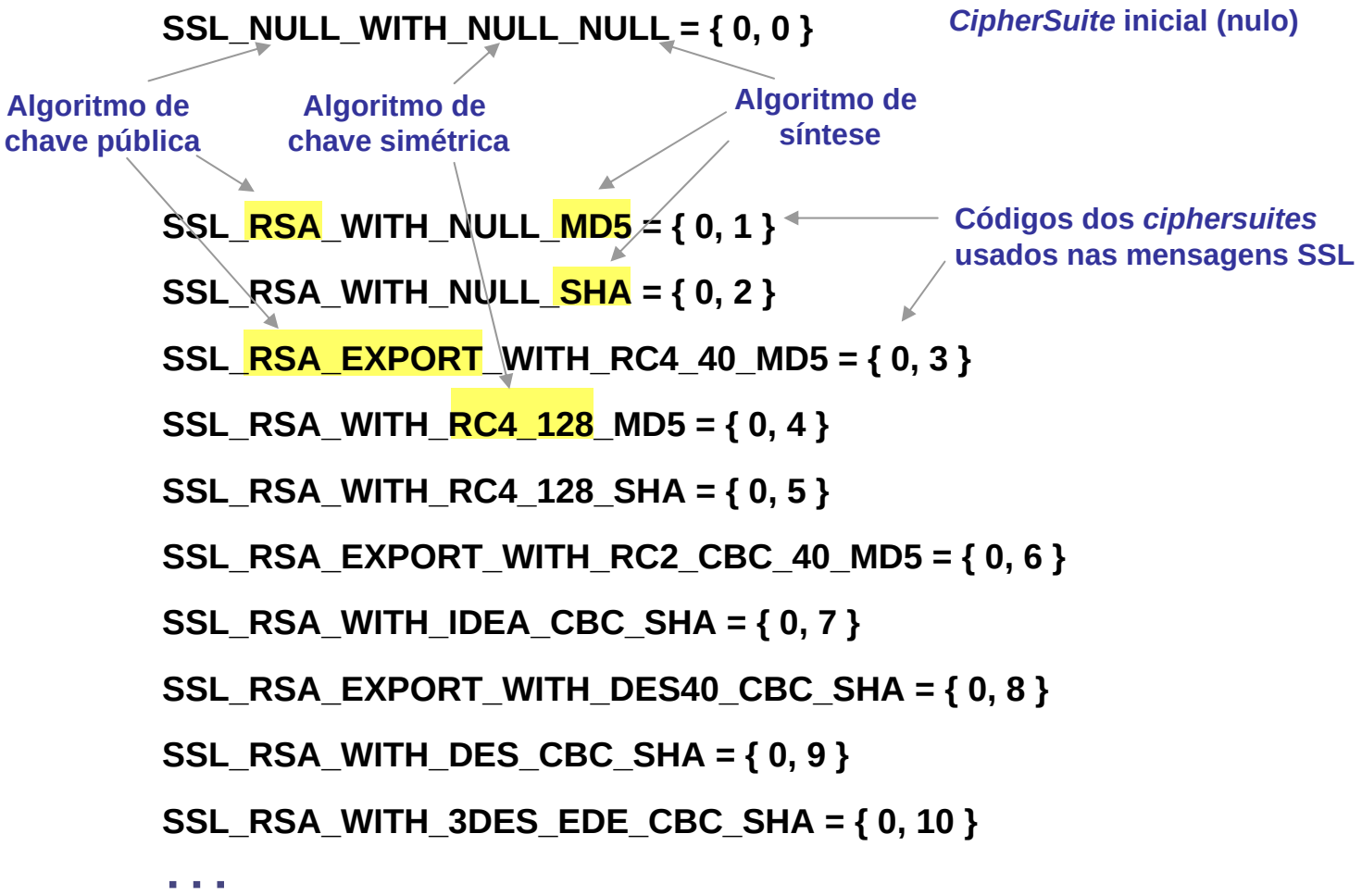
Mensagem ClientHello

CipherSuite

- *KeyExchange* (esquema para troca de chave simétrica)
  - *RSA (chave de sessão é cifrada com chave pública do servidor): mais comum*
  - *Fixed Diffie-hellman (certificado de chave pública DH)*
  - *Ephemeral Diffie-Hellman (chave pública DH assinada com RSA ou DSS)*
  - *Anonymous Diffie-Hellman (sem certificados → vulnerável a ataque MIM)*
- *CipherSpec* (algoritmos simétricos e de síntese, para confidencialidade, autenticidade e integridade)
  - *CipherAlgorithm: RC4, RC2, DES, 3DES…*
  - *MACAlgorithm: MD5 ou SHA1*
  - *IsExportable*
  - *HashSize*
  - *Key Material*
  - *IV Size*

Mensagem ClientHello

CipherSuite



Mensagem ServerHello

- Versão do protocolo
- Número aleatório
  - Previne ataques de repetição
- ID da sessão
  - Permite ao cliente retomar a sessão mais tarde
- *CipherSuite*
  - Normalmente, mas não obrigatoriamente, a escolha recai na proposta mais segura
- Compression method

Mensagem Certificate

- Utilizada para transportar certificados
- Opcional
  - Mas, sem certificados → vulnerável a ataque "*Man in the Middle*"
- Normalmente certificados X.509
  - Corrente de certificados: certificado do servidor, da CA…
- Tipicamente, só o servidor apresenta certificado
  - Devido essencialmente ao custo dos certificados e sua gestão
  - Cliente (utilizador) pode autenticar-se ao nível do protocolo de aplicação
    - *Username/password*
- Certificado X.509 associa uma chave pública a uma identidade

Mensagem Certificate

- Os certificados devem ser verificados
- A CA tem de ser confiada por ambos, servidor e cliente
  - Uma CA pode gerar um certificado para uma outra CA (sub-CA)
- É necessário verificar que o certificado não foi revogado
  - *Certificate Revocation List* (CRL)
  - *Online Certificate Status Protocol* (OCSP)

Mensagem ClientKeyExchange

- PremasterSecret
  - Usada como uma semente para calcular outras chaves de sessão
  - Quando *KeyExchange=RSA:*
    - *Chave simétrica criada pelo cliente*
    - *2 bytes com a versão do SSL + 46 bytes aleatórios*
    - *Chave é cifrada com chave pública do servidor (RSA) e enviada para o servidor*

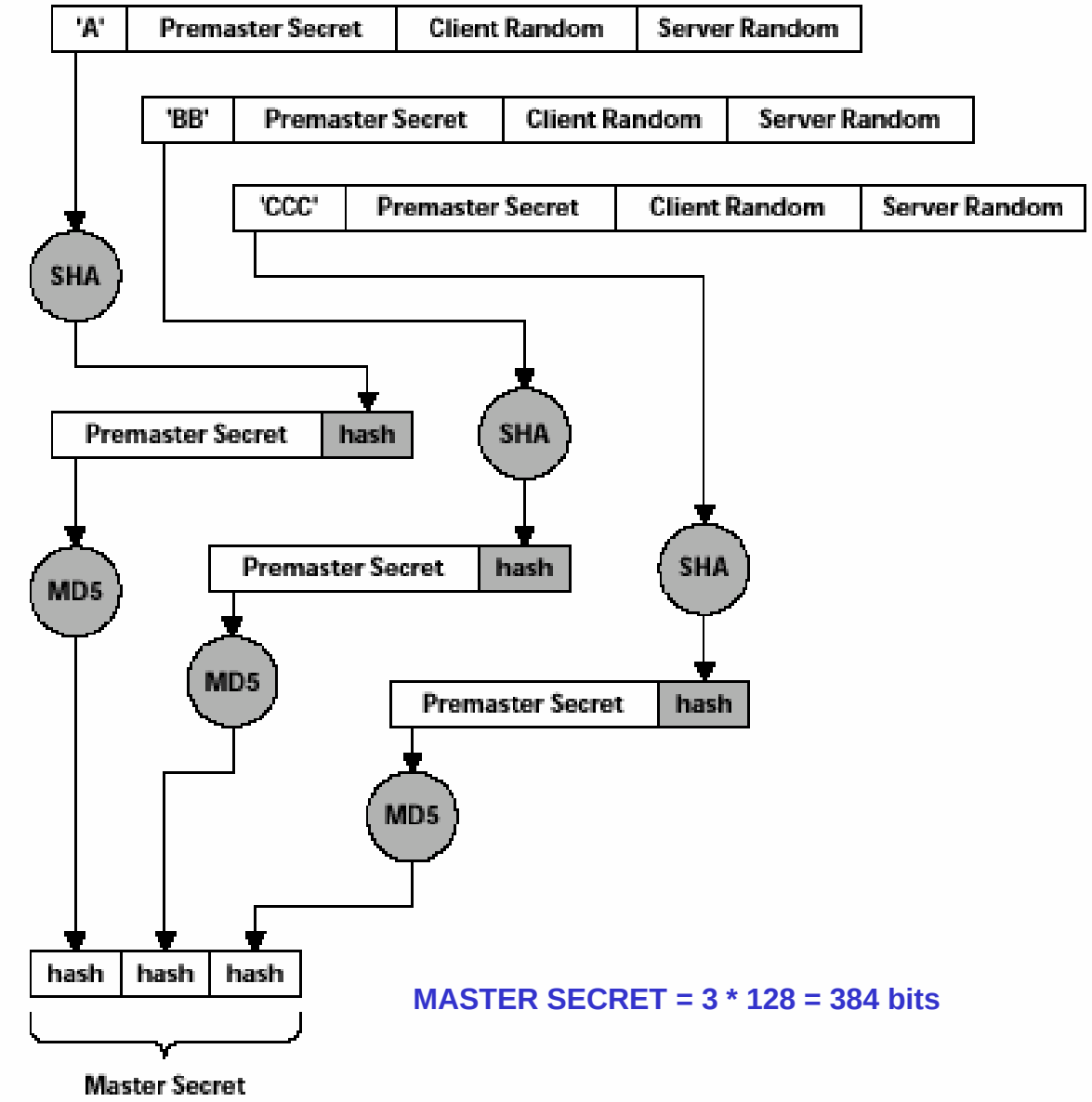
Mensagens ChangeCipherSpec e Finished

- Change Cipher Spec
  - Aplica os algoritmos e respectivas chaves de sessão
  - A partir daqui os dados são cifrados
- Finished
  - Primeira mensagem cifrada
  - Finaliza o processo de handshake

Geração de chaves

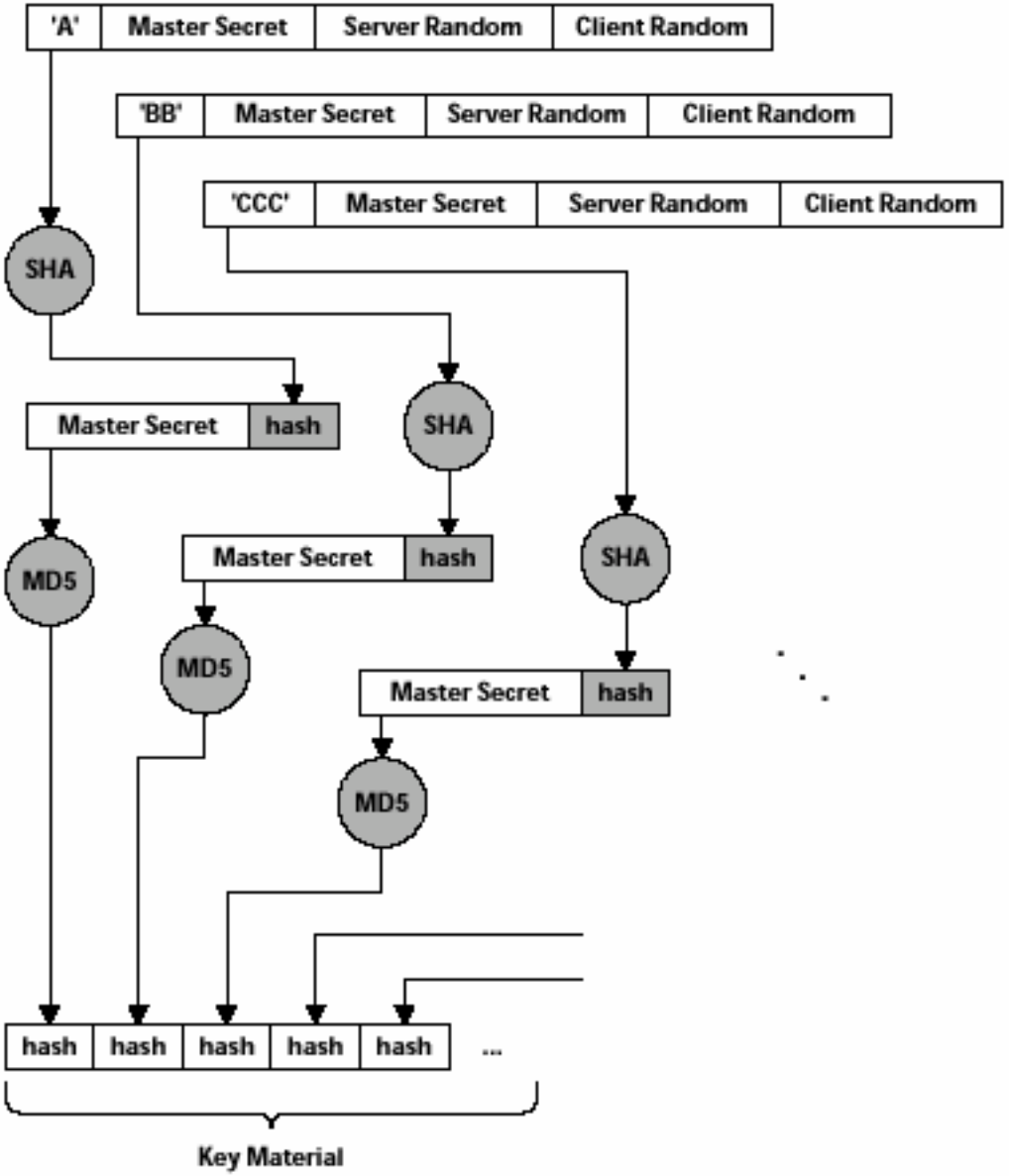
- PremasterSecret
  - Chave simétrica criada pelo cliente
- MasterSecret
  - Calculada por ambos, servidor e cliente, a partir da chave PremasterSecret e dos números aleatórios gerados pelo cliente e pelo servidor.
- Material criptográfico (*Key Material*)
  - Gerado a partir da chave MasterSecret e dos números aleatórios
- Chaves de sessão
  - Extraídas a partir do material criptográfico para
    - *autenticação e integridade das mensagens (MAC)*
    - *confidencialidade – cifragem dos dados (ex: DES, RC4)*
    - *vectores de inicialização – algoritmos simétricos por bloco (ex: DES) em modo CBC ou algoritmos simétricos por fluxo (ex: RC4)*
  - Um conjunto de chaves para cada sentido

MasterSecret

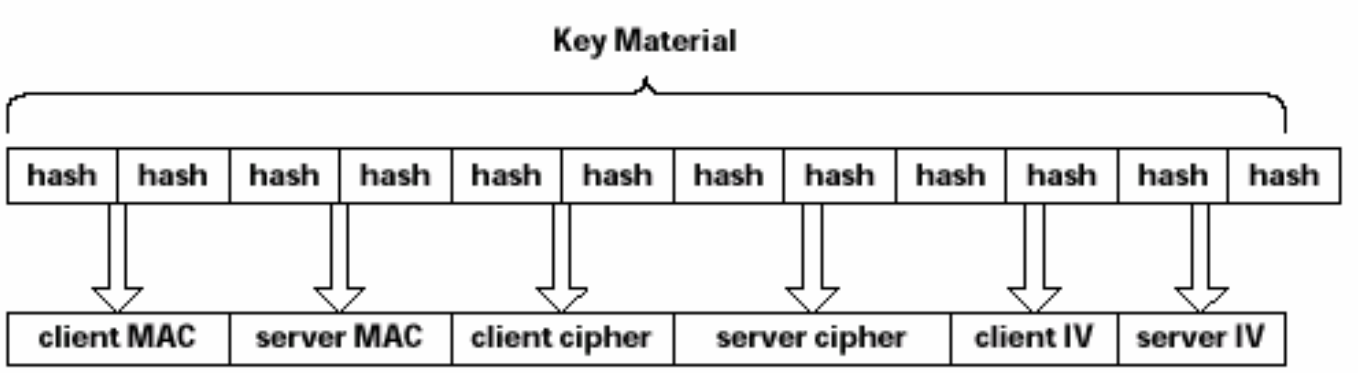


MASTER SECRET = 3 * 128 = 384 bits

Material criptográfico



Chaves de sessão



Sessões

- Sessões vs. conexões
  - Múltiplas conexões na mesma sessão
  - Uma conexão TCP → uma conexão SSL
  - Uma negociação por sessão
- Retoma de sessão
  - Através de IDs de sessão
  - Cliente usa o FQDN ou endereço IP como referência
  - O servidor usa o ID de sessão fornecido pelos clientes
- Repetição de *handshake*
  - O cliente pode iniciar um novo *handshake* durante a mesma sessão

Overhead

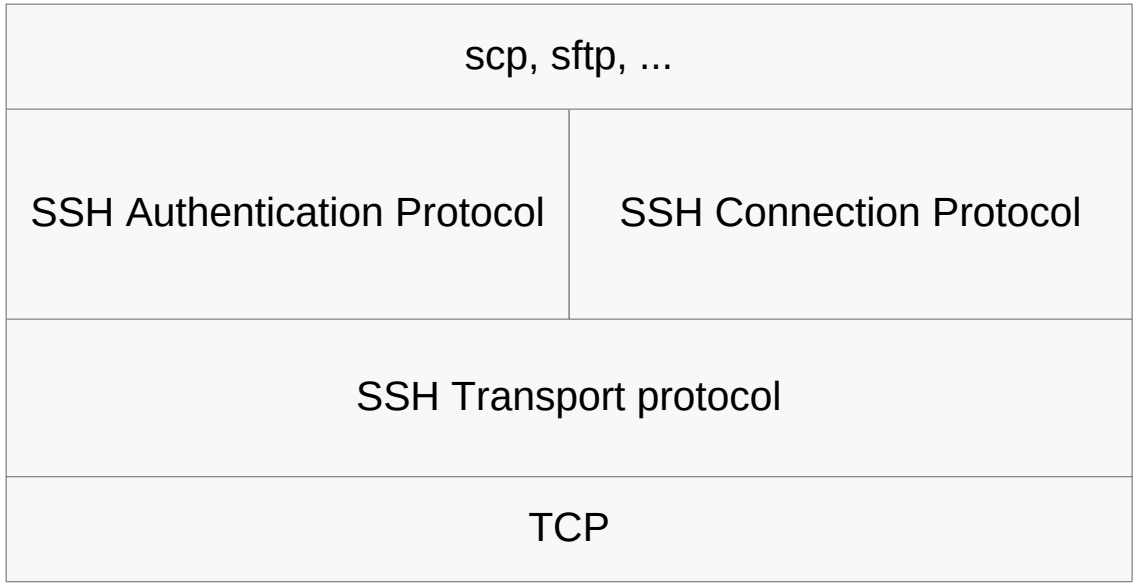
- 2 a 10 vezes mais lento que uma sessão TCP
- Atraso devido a:
  - *Handshake*
    - *Recurso a criptografia de chave-pública*
  - Transferência de dados
    - *Dados são cifrados com criptografia simétrica*

SSH – Secure SHell

Enquadramento

- Objectivo inicial: substituir o telnet, rlogin, rsh… por uma alternativa segura
- Actualmente também permite
  - Substituição do FTP e rcp por sftp/scp para transferência segura de ficheiros
  - Encapsulamento seguro de protocolos de aplicação (ex: POP3, SMTP)
- SSH 1 desenvolvido por Ylönen em 1995
- IETF ➔ SSH 2
- Arquitecturas
  - SSH 1 ➔ monolítica
  - SSH 2 ➔ arquitectura melhorada ➔ vários sub-protocolos
- SSH1 ➔ vulnerável a ataques MIM

Pilha protocolar



Pilha protocolar

- *Transport Protocol*
  - Autenticação do servidor
  - Estabelecimento de um canal seguro para transporte dos outros protocolos
  - Compressão opcional dos dados
  - Transportado sobre protocolo de transporte fiável → TCP
- *Authentication Protocol*
  - Autentica o cliente
  - Diversos protocolos de autenticação
- *Connection Protocol*
  - Permite o suporte de várias sessões em simultâneo. Cada sessão é encapsulada num canal lógico.

Transport Layer protocol

- Negociação de:
  - Método para troca de chaves de sessão
  - Algoritmo de chave pública para autenticação do servidor
  - Algoritmo de chave simétrica para confidencialidade
  - Algoritmo MAC para integridade e autenticidade
- Confidencialidade
  - Cada pacote é cifrado com um algoritmo de chave simétrica
  - Algoritmos: *3des-cbc, blowfish-cbc, twofish256-cbc, twofish192-cbc, twofish128-cbc, aes256-cbc, aes192-cbc, aes128-cbc, …, nenhum (não recomendado)*
- Integridade e autenticidade
  - A cada pacote é acrescentado um MAC (*Message Authentication Code*)
  - Algoritmos: *hmac-sha1, hmac-sha1-96, hmac-md5, hmac-md5-96, nenhum (não aconselhável)*
- Compressão dos dados opcional: ZLIB (LZ77)

Troca de chaves e autenticação do servidor

- Recorrendo ao método Diffie-Hellman, cliente e servidor geram um chave simétrica de sessão K comum, a qual servirá para derivar novas chaves simétricas para confidencialidade e integridade (e autenticidade): uma chave para confidencialidade, outra para integridade, em cada sentido.
  - Método único para derivação de K: *diffie-hellman-group1-sha1*
- Cada servidor tem um par de chaves assimétricas. A pública (*Host Key*) é utilizada para autenticar o servidor perante o cliente.
- Servidor autentica-se enviando a sua chave pública (ou certificado com a mesma) juntamente com o resumo da chave K cifrado com a sua chave privada (assinatura).
- Cliente verifica assinatura → confirma que o servidor é o detentor da chave privada correspondente e que não houve um ataque MIM

Validação da chave pública

- Dois métodos para a validação:
  - Através de um certificado (ex: pgp, x.509, spki)
  - Cliente gere uma base de dados do tipo {host/chave}
    - *Cada vez que se inicia uma sessão, se servidor novo ou se chave modificada → cliente ssh informa utilizador do fingerprint (resumo) da chave*
    - *Decisão de confiança é do utilizador → quando não verificado é possível um ataque MIM*
    - *Solução para evitar/minimizar ataques MIM: divulgação do fingerprint através de canais alternativos (ex: telefone, e-mail).*
- Algoritmos: ssh-dss, ssh-rsa, x509v3-sign-rsa, x509v3-sign-dss, spki-sign-rsa, spki-sign-dss, pgp-sign-rsa, pgp-sign-dss

Autentication Protocol

- Autentica o utilizador perante o servidor
- Métodos suportados
  - ***password*** – UserID/*Password* (o mais simples e mais vulgar)
  - **pubkey** – UserID/chave pública do utilizador
  - **hostbased** – (UserID/)chave pública do *host*
  - Outros métodos
    - *Certificado digital*
    - *S/Key*
    - *SecurID*
    - *Kerberos*
    - …
- Também permite a alteração de *password*

Connection Protocol

- Oferece os serviços:
  - Sessão remota interactiva
  - Execução remota de comandos
  - Reencaminhamento de conexões TCP
  - Reencaminhamento de conexões X11
- Para cada serviço são estabelecidos um ou mais canais lógicos
- Todos os canais lógicos são multiplexados numa conexão *Transport Protocol*

Connection Protocol

Reencaminhamento de conexões TCP

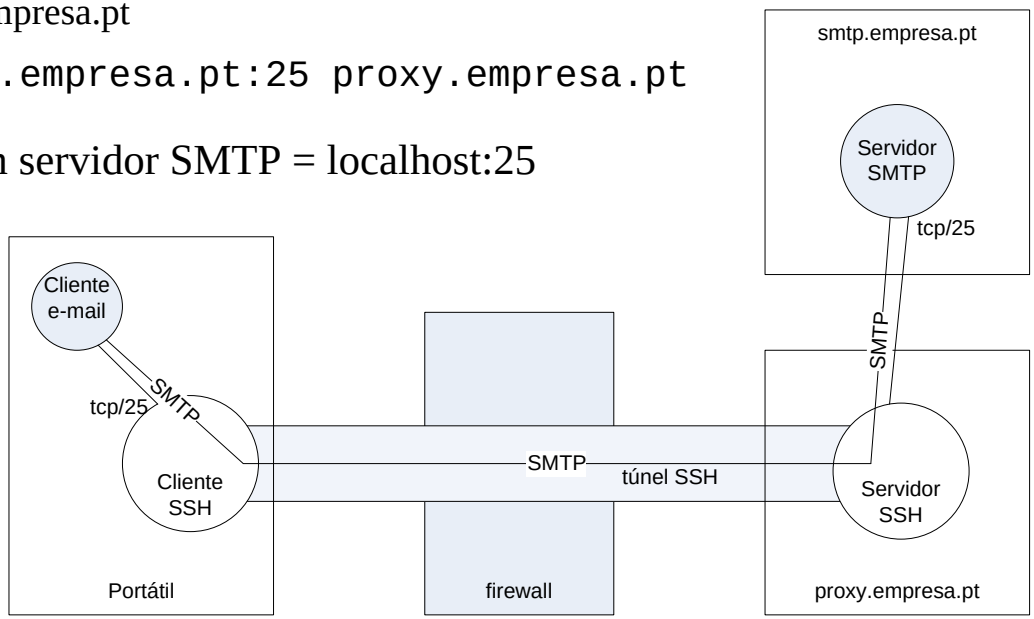
- Permite ultrapassar firewalls
- Criar túneis seguros
- Parâmetros
  - `Porto local` ➔ porto onde cliente ssh fica à escuta de conexões TCP
  - (`host local`)
  - `Host remoto` ➔ *host* onde reside o serviço (aplicação remota)
  - `Porto remoto` ➔ porto no *host* remoto onde o serviço está à escuta
  - `Servidor SSH` ➔ *host* que corre o servidor SSH

Reencaminhamento de conexões TCP

- Fluxo de dados
  - Aplicação cliente estabelece uma conexão TCP para `host local/porto local`
  - Cliente SSH encapsula dados transferidos na conexão tcp para um canal seguro entre `host local` e `servidor SSH`
  - Quando chegam a `servidor SSH`, dados são novamente encapsulados numa conexão TCP (em claro) entre `servidor SSH/porto dinâmico` e `host remoto/porto remoto`

Reencaminhamento de conexões TCP: Exemplo

- Envio de e-mails a partir de portátil quando fora da rede da empresa
- Configurar cliente SSH
  - `Porto local` = 25
  - `Host remoto` = smtp.empresa.pt
  - `Porto remoto` = 25
  - `Servidor SSH` = proxy.empresa.pt
  - `port# ssh -L 25:smtp.empresa.pt:25 proxy.empresa.pt`
- Configurar cliente de e-mail com servidor SMTP = localhost:25



SSH 1 – Vulnerabilidades

- SSH 1 não é seguro, tem várias vulnerabilidades
- Por questões de compatibilidade clientes e servidores vêm por vezes com suporte de SSH 1 activado
- Mesmo que cliente e servidor sejam configurados para usar SSH 2 por omissão, é possível que um atacante altere as mensagens iniciais entre cliente e servidor em que acordam a versão a utilizar de forma a que seja usado o SSH 1.
- Depois explora vulnerabilidades do SSH 1 → MIM