PMR3412 - Redes Industriais - 2023

Aula 11 - Segurança: Certificados X.509 e Transport Layer Security (TLS)

Prof. Dr. Newton Maruyama

9 de Novembro de 2023

PMR-EPUSP

Notas de autoria (Disclaimer) - Equipe envolvida

Os slides que serão utilizados nesse ano são baseados no curso desenvolvido para os anos 2020, 2021 e 2022. Participaram da concepção do curso e desenvolvimento do material os seguintes professores:

- Prof. Dr. André Kubagawa Sato
- Prof. Dr. Marcos de Sales Guerra Tsuzuki
- Prof. Dr. Edson Kenji Ueda
- Prof. Dr. Agesinaldo Matos Silva Junior
- Prof. Dr. André César Martins Cavalheiro

Conteúdo

1. Revisão

2. Combinando Algoritmos Simétricos e Assimétricos

3. Certificados X.509

4. Transport Layer Security (TLS)

5. Referências

Revisão

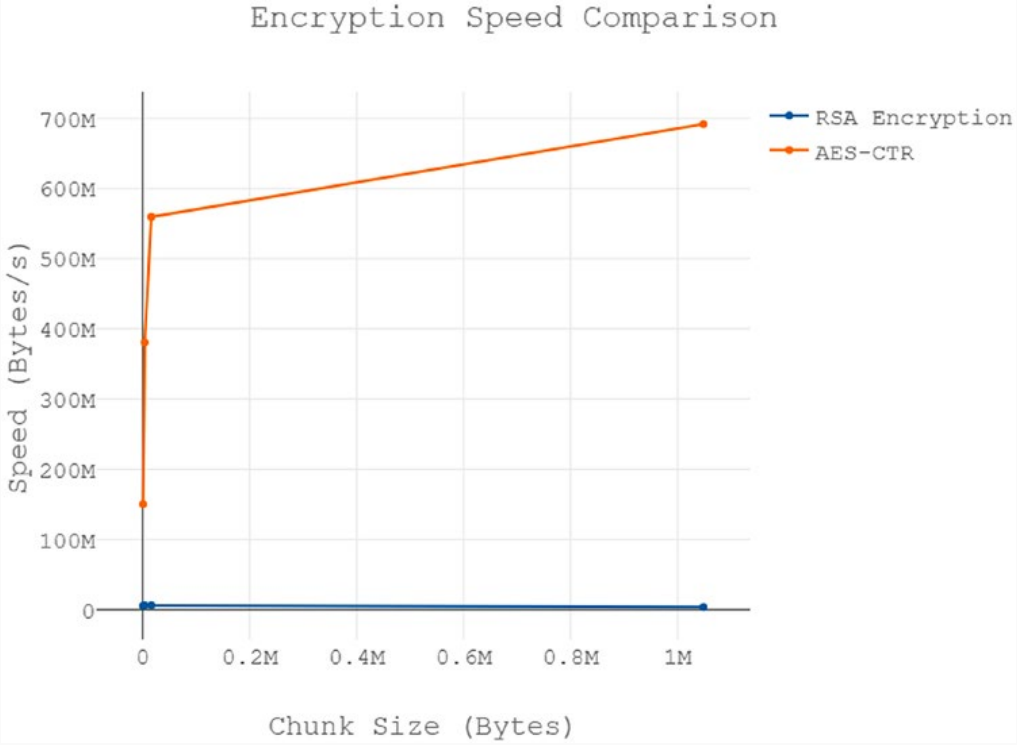
Revisão - Princípios básicos

- Princípio de Kerckhoff: **um sistema criptográfico deve ser seguro mesmo se tudo é conhecido sobre ele, exceto a chave**.
- A criptografia é a principal ferramenta para providenciar proteção para informação. Ela fornece as seguintes proteções:

- Duas estratégias: criptografia simétrica e criptografia assimétrica

Revisão - Criptografia simétrica × Criptografia assimétrica

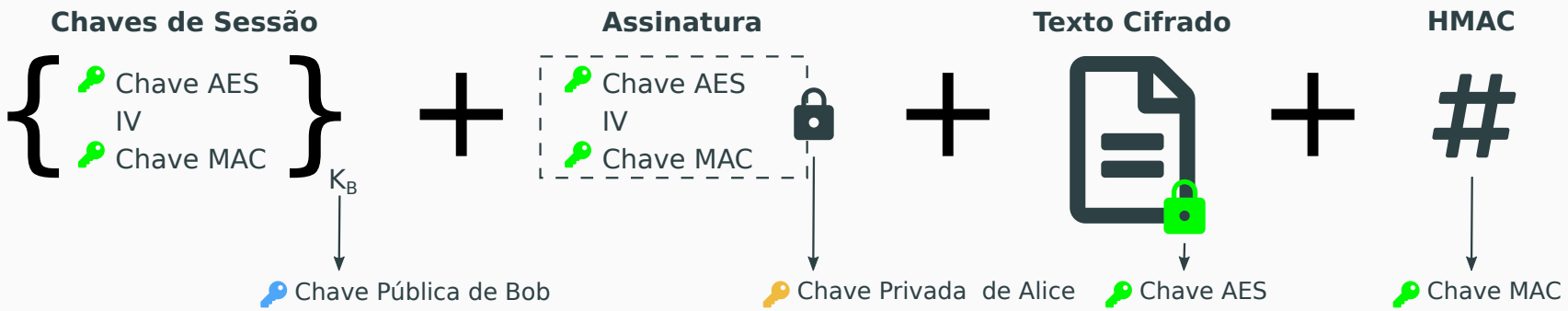
- **Confidencialidade**: encriptação (ambos)
- **Autenticação e integridade**:
  - criptografia simétrica: HMAC
  - criptografia assimétrica: assinatura + certificado
- **Performance**:



Combinando Algoritmos Simétricos e Assimétricos

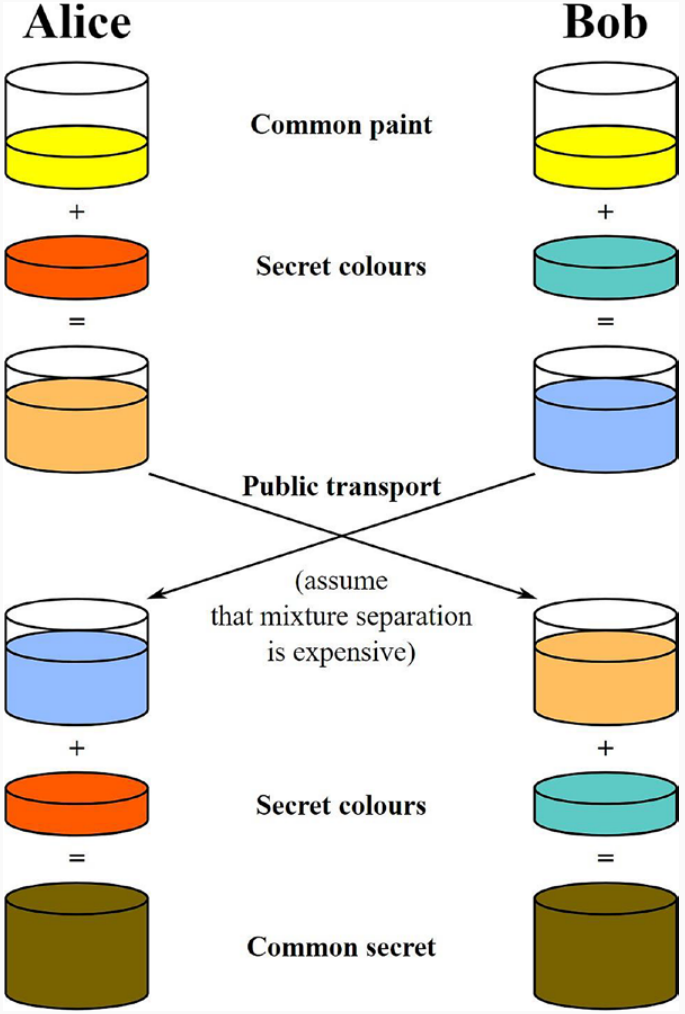
- Assumindo que as chaves públicas e certificados já estão em posse das pessoas envolvidas: Alice e Bob.
- Algoritmo simplificado: uma transmissão de Alice para Bob pode ser um stream de bytes concatenadas contendo:

Dado	Chave	Encriptado?
Chave AES, IV e chave MAC	Chave pública de Bob	Sim
Assinatura de Alice da chave AES , IV, chave MAC	Chave privada de Alice	Não
Mensagem	Chave AES	Sim
HMAC	Chave HMAC	Não



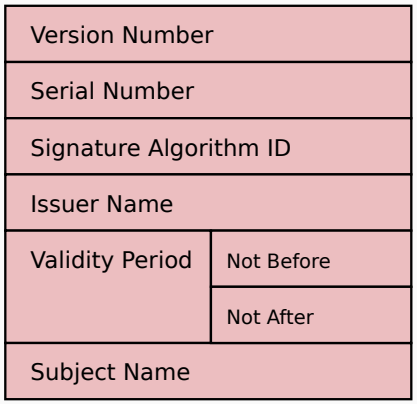
Combinando Algoritmos Simétricos e Assimétricos - Troca de Chaves com Diffie-Hellman

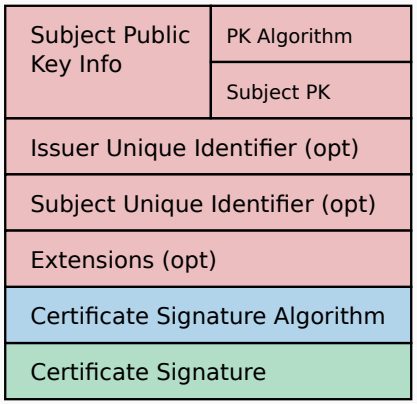
- Diffie-Hellman (DH) ou seu variante Elliptic-Curve Diffie-Hellman (ECDH) são algoritmos de criptografia assimétricas utilizados apenas para a troca de chaves.
- Diferentemente do RSA, a troca de chaves com DH não envolve troca de nenhum "segredo", encriptado ou não.
- Simplificando bastante, o DH/ECDH funciona da seguinte forma:
  1. cada pessoa inicialmente possui uma chave pública e uma privada;
  2. as partes fazem a troca de chaves públicas; e
  3. ao combinar a chave público do outro com a sua chave privada, é criado um "segredo compartilhado" (geralmente a chave simétrica).



Certificados X.509

- O certificado X.509 é o tipo mais comum de certificado utilizado atualmente; é o certificado adotado no protocolo TLS (*Transport layer Security*).
- Consiste em um conjunto de pares chave-valor, com possibilidade de subcampos.
- Os *Certificate Signing Requests* (CSRs) devem ser criados para requisitar um certificado a um *Certificate Authority*.
- Os CSRs possuem o mesmo formato que o certificado X.509, com alguns campos vazios, como por exemplo o *Issuer Name* (Obviamente o issuer name aparece somente no certificado).





Issuer Name / Subject Name:

CN: CommonName
OU: OrganizationalUnit
O: Organization
L: Locality
S: StateOrProvinceName
C: CountryName

Certificados X.509 - Demonstração com OpenSSL

- O OpenSSL pode ser utilizado para gerar chaves, criar CSR e, finalmente, o certificado X.509 (possivelmente auto-assinado).
- Para criar uma chave RSA privada, podemos executar:

```
genpkey -algorithm RSA -out domain_key.pem -pkeyopt rsa_keygen_bits:2048
```

- Para gerar o CSR, que extrai a chave pública também, podemos executar:

```
openssl req -new -key domain_key.pem -out domain_request.csr
```

- Para gerar o certificado auto-assinado, podemos executar:

```
openssl x509 -req -days 30 -in domain_request.csr -signkey domain_key.pem -out domain_cert.crt
```

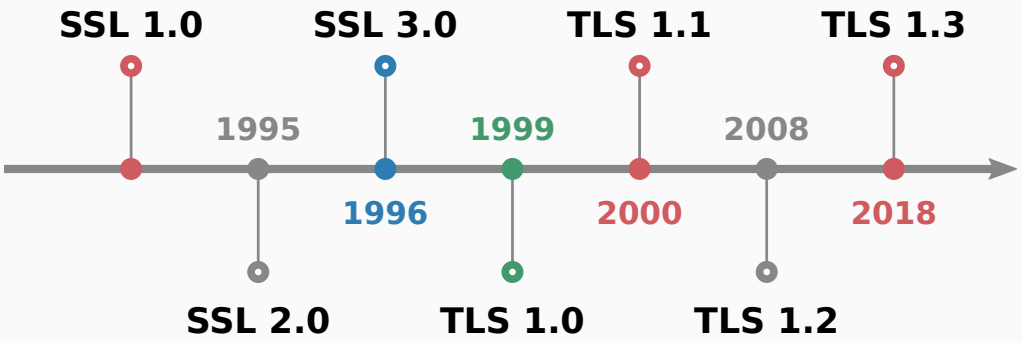
- Para testar, podemos inicializa um servidor com:

```
openssl s_server -accept 8888 -www -cert domain_cert.crt -key domain_key.pem
```

Transport Layer Security (TLS)

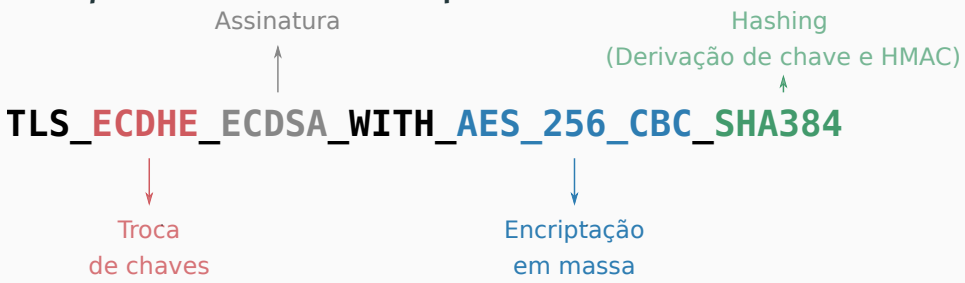
TLS - Introdução

- **Objetivo:** adicionar uma camada de segurança de transporte (confidencialidade e autenticação) ao TCP/IP. Como vimos no curso, o protocolo TCP/IP não possui nenhuma garantia de segurança.
- Brevíssimo histórico:
  - surgiu como Secure Sockets Layer (SSL) com o Netscape nos anos 90;
  - seguiram os SSL2 e SSL3, quando foi renomeada para TLS 1.0;
  - atualmente as versões TLS 1.2 e TLS 1.3 são utilizadas.
- **Ponto crítico:** Handshake; pois estabelece identidade e deriva as chaves de sessão para o transporte seguro.

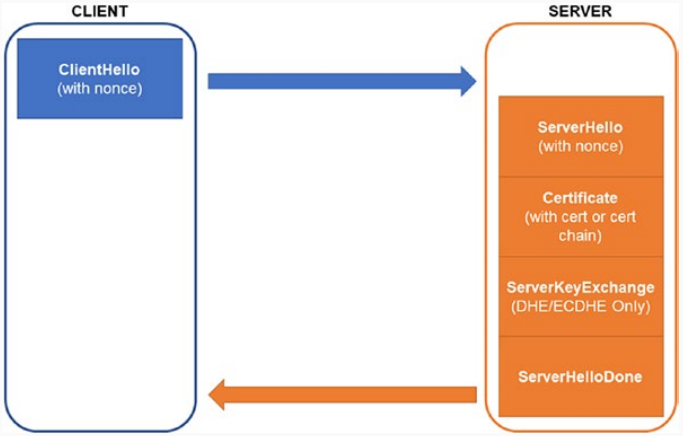


TLS - *Cipher Suites* e "Hellos" Introdutórios (TLS 1.2)

- O TLS consiste de uma combinação de protocolos que trabalham juntos, estes são definidos nos *cipher suites*. Exemplo:



- O TLS 1.2 se inicia com a mensagem de "Hello" do cliente, que envia um nonce[1] e as configurações TLS (incluindo lista de *cipher suites* suportados).
- O servidor responde com certificados e informações para troca de chaves (opcionalmente).



[1] número aleatório que só pode ser utilizado uma única vez numa comunicação criptográfica.

TLS - Autenticação do Cliente e Derivação das Chaves de Sessão

Autenticação do cliente:

- Como vimos no exemplo do "Hello", apenas o servidor é autenticado (servidor envia o certificado para o cliente).
- Essa é a configuração padrão, uma vez que é suficiente para servidores na internet cujo objetivo é propagar a informação.
- Nos casos em que o servidor deve autenticar o cliente, isso é geralmente feito com nome de usuário e senha.

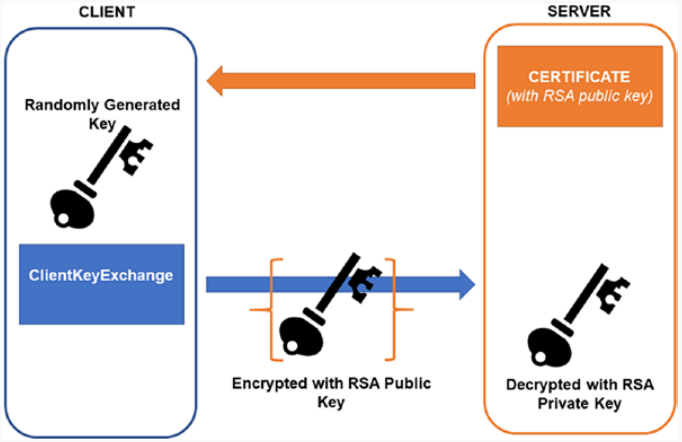
Derivando chaves de sessão:

- Existem duas formas de trocar chaves simétricas: transporte de chaves e concordância de chaves.
- O objetivo do handshake do TLS 1.2 é obter o "pre-master secret" (PMS), tanto no cliente como no servidor.
- O PMS é utilizada para gerar o "master secret" que, por sua vez, gera as chaves de sessão.

TLS - Troca de chaves: RSA vs DHE/ECDHE

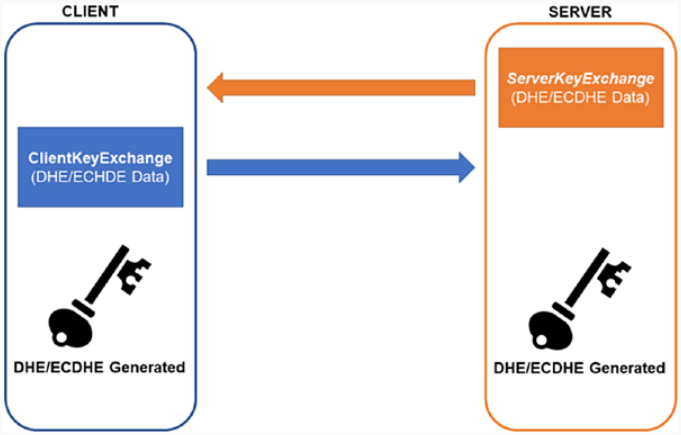
RSA:

- Server "Hello" não envia nenhum parâmetro.
- Cliente encripta o PMS com a chave pública do servidor e a envia. Não é necessário assinatura.
- Desvantagens: PMS é gerado inteiramente pelo cliente, padding PKCS 1.5 é vulnerável.



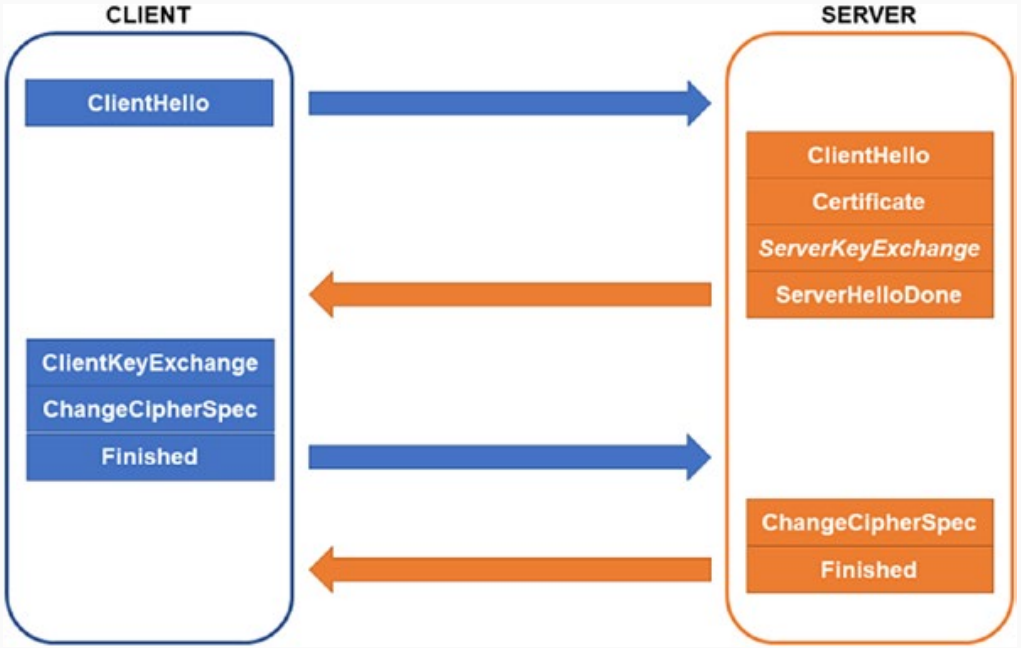
DHE/ECDHE:

- Servidor envia a chave pública DHE ou ECDHE, que é efêmera, e seus parâmetros.
- A chave RSA/ECDSA privada é utilizada para assinatura destes.
- Vantagem: *forward secrecy*, a chave não revela informações sobre as comunicações prévias.



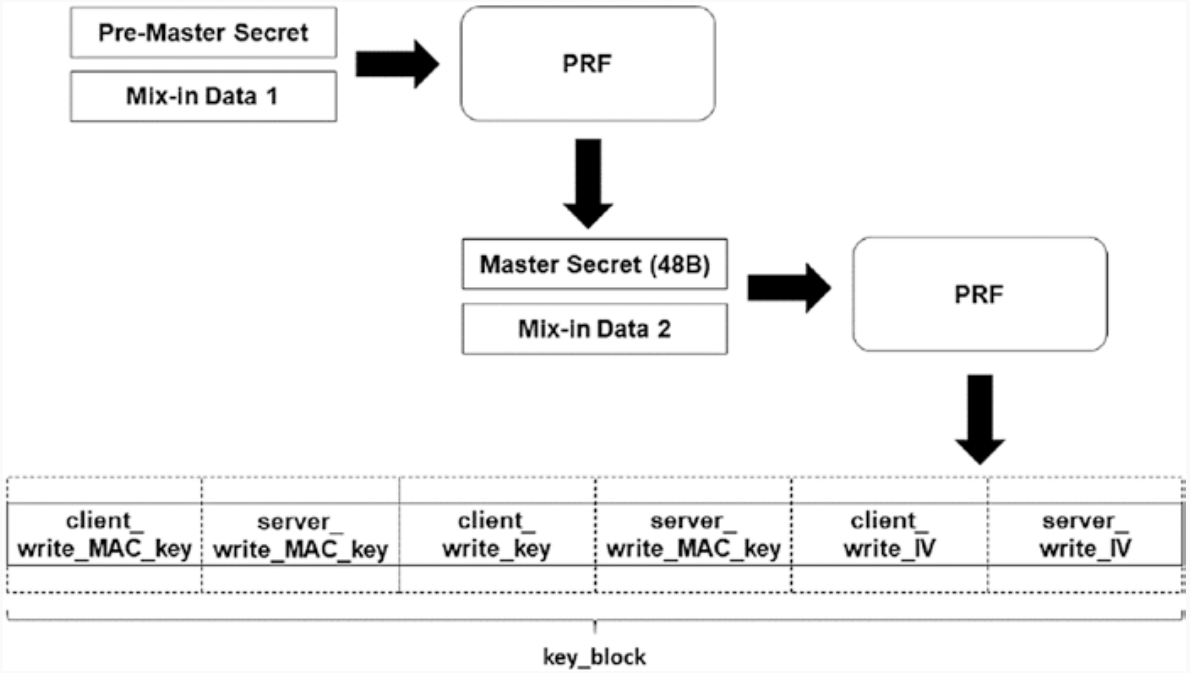
TLS - Mudando para a nova Cifra

- Após a finalização de troca de chaves, toda comunicação deve ser encriptada e autenticada.
- Para finalizar o *Handshake* e mudar para a cifra de transferência em massa, faltam os seguintes passos:
  1. Cliente e Servidor enviam mensagens `ChangeCipherSpec` para indicar que comunicação será encriptada partir de agora.
  2. Cliente e Servidor enviam mensagens `Finished` para completar o Handshake.



TLS - Derivando Chaves

- Após o *handshake*, o cliente já verificou a identidade do servidor através do certificado e ambos os lados possuem o PMS (pre-master secret).
- Para gerar o "master secret", o algoritmo de hash é utilizado para expandir o PMS para 48 bytes.
- O "master secret" é, então, expandido para gerar as chaves, ou `key_block`.
- O `key_block` pode conter até seis parâmetros: chave MAC de escrita do cliente, chave MAC de escrita do servidor, chave de escrita do cliente, chave de escrita do servidor, IV de escrita cliente, IV de escrita servidor.

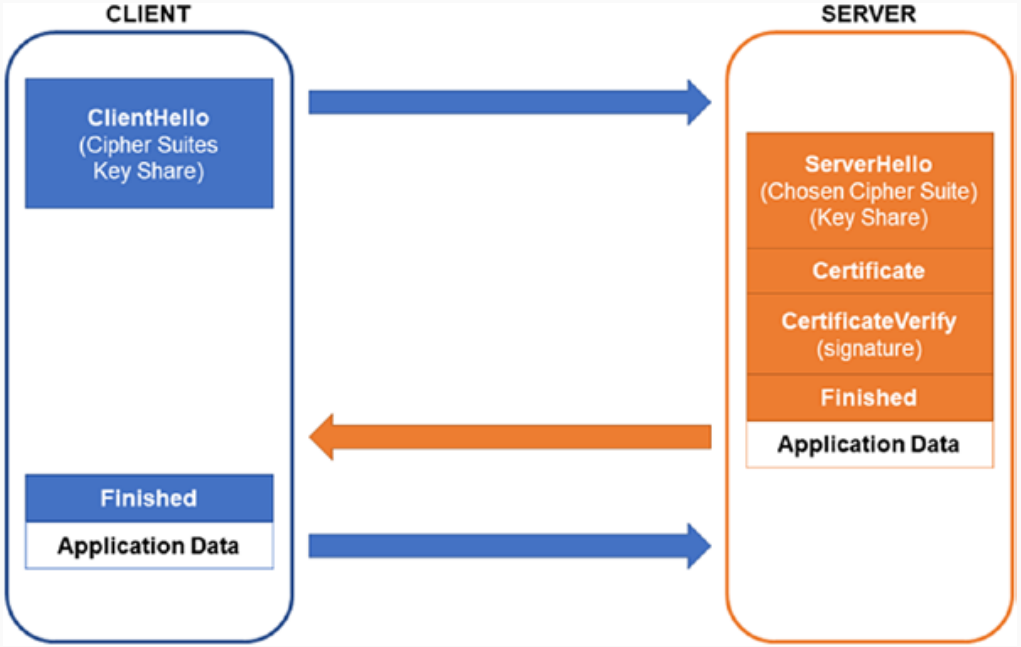


TLS - Transferência de Dados

- Com as chaves simétricas geradas, é possível realizar a transferência em massa (*bulk transfer*).
- O TLS transmite os dados em uma estrutura de dados chamada `TLSCipherText`, que comporta até 16K de dados.
- Utilizando a linguagem C, podemos descrever esta estrutura como:

```
struct {
    ContentType type;
    ProtocolVersion version;
    uint16 length;
    select (SecurityParameters.cipher_type) {
        case stream: GenericStreamCipher; // inclui MAC
        case block: GenericBlockCipher; // inclui MAC
        case aead: GenericAEADCipher; // inclui MAC
    } fragment;
} TLSCiphertext;
```

- Na versão TLS 1.3 foram removidos diversos *cipher suites*, inclusive o suporte a RSA para troca de chaves.
- A modificação mais significativa foi a redução do tempo de latência para o *Handshake*, que ocorre apenas com uma única rodada de troca de mensagens.
- Esta modificação é bastante importante para protocolos sem estado como o HTTP. Nesses casos, abrir um túnel TLS 1.2 para cada conexão é bastante custoso.



Referências

Referências - Aula 11

- Capítulos 8 do livro "Practical Cryptography in Python: Learning Correct Cryptography by Example", Seth James Nielson e Christopher K. Monson, Apress, 2019.

The End!