Existem 3 propriedades principais: confidencialidade, integridade (autenticidade e não-repúdio) e disponibilidade.

1. **Confidencialidade**: Impedir a divulgação de informações não autorizadas, ou seja, conteúdo de utilizadores não autorizados e a informação não pode ser vista nem analisada.
   Privacidade: Inclui meios para garantir quais são as informações que podem ser vistas e por quem podem ser vistas.
2. **Integridade**: Garantir que as informações/dados recebidos são exatamente iguais aos dados enviados por uma entidade autorizada. O conteúdo não pode ser modificado, corrompido ou perdido por terceiros. Garante autentidade, ou seja, garante que a entidade envolvida é, de facto, aquela que afirma ser. Inclui também evitar que entidades se recusem a gerar informações (não-repúdio).
3. **Disponibilidade**: A informação é acessível e utilizável sob demanda por uma entidade autorizada.

Proteção de informação: A informação e os dados estão em dispositivos de armazenamento e redes de computadores. A proteção pode estar em diferentes níveis:

- **Hardware**: dispositivos de processamento, armazenamento, …
- **Software**: sistema operacional, aplicações, bibliotecas, …
- **Dados**: ficheiros, databases, passwords, …
- **Comunicação**: routers, …

Exemplos de ataques

Existem dois tipos possíveis de ataques: ataques passivos (interceção) e ataques ativos (interrupção, modificação e fabricação).

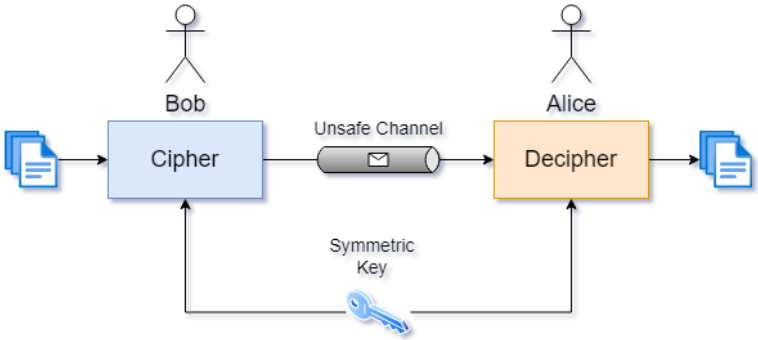
Ataques passivos são ataques de divulgação de conteúdo (o invasor lê o conteúdo da informação) e ataques de análise de tráfego (o invasor analisa o tráfego para extrair informações).

Ataques ativos são ataques máscara (o invasor faz-se passar por uma entidade autorizada), ataques replay (o invasor captura uma unidade de dados e reproduz essa mesma unidade para produzir uma ação autorizada), ataques de modificação da mensagem (o invasor modifica o conteúdo de uma unidade de dados) e ataques de navegação de serviço (o invasor impede que a informação esteja disponível).

Esquemas Simétricos

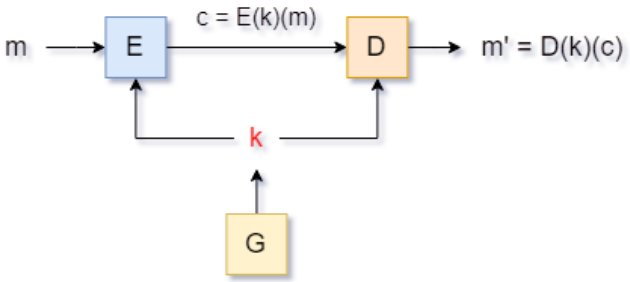
- O processo de proteção e desproteção utiliza a mesma chave.
- As chaves geralmente são usadas durante um curto período de tempo e são estabelecidas após um processo de negociação entre quem criptografa e quem descriptografa.
- Os esquemas simétricos são usados em cifras simétricas e esquemas MAC.

1. Cifras Simétricas



Algoritmos (G, E, D):

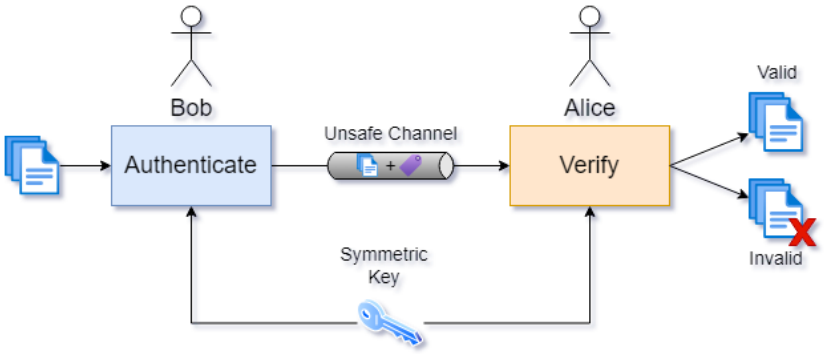
G	função probabilistica para geração de chaves	G :-> Keys
E	função probabilistica para cifra	E: Keys -> {0, 1}* -> {0, 1}*
D	função deterministica para decifrar	D: Keys -> {0, 1}* -> {0, 1}*



Propriedades

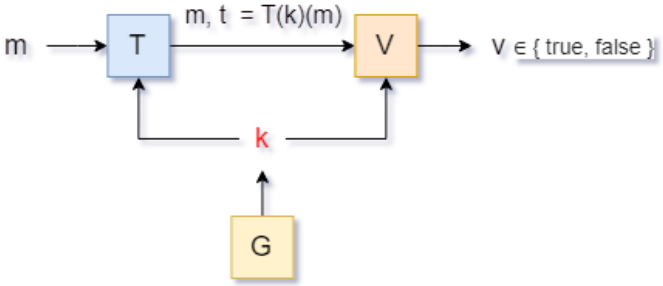
- Keys: D(k)(E(k)(m)) = m
- Propriedade de segurança: é computacionalmente inviável encontrar m (texto simples) a partir de c (texto cifrado), sem conhecer a chave k
- Esquema simétrico: utilização da mesma chave k nas funções E e D
- A mensagem m e o texto cifrado c são sequências de bytes com dimensão variável ({0, 1}*)
- Não garante a integridade dos dados
- Exemplos: DES-CBC-PKCS5Padding
  AES-CBC-PKCS5Padding

2. Esquemas MAC



Algoritmos (G, T, V):

G	função probabilistica para geração de chaves	G :-> Keys
T	função probabilistica para geração de tags	T: Keys -> {0, 1}* -> Tags
V	função deterministica para verificação de tags	V: Keys -> (Tags x {0, 1}*) -> {true, false}

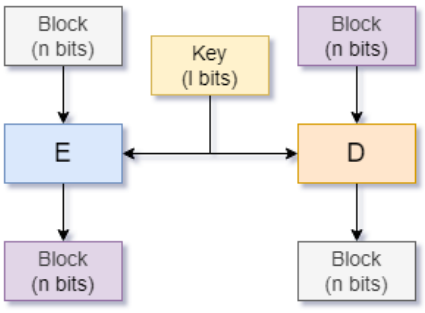


Propriedades

- Keys: V(k)(T(k)(m), m) = true
- Propriedade de segurança: é computacionalmente inviável encontrar m (texto simples) a partir de t (tag), sem conhecer a chave k
- Esquema Simétrico: utilização da mesma chave k nas funções T e V
- A mensagem m e a tag t são sequências de bytes com dimensão variável ({0, 1}*)
- A tag t tem dimensão fixa (por exemplo 160, 256 bits)
- Detetores de código e corretores de erros não servem para esquemas MAC
- Exemplos: HMAC-SHA1
  HMAC-SHA256

3. Primitivas de cifra simétrica

Para usar esquemas de cifra simétrica, precisamos de escolher uma primitiva de cifra simétrica (exemplos: DES, AES, Blowfish, etc...).
Primitivas de cifras de bloco : n (dimensão do bloco) e l (dimensão da key).
Função E: {0, 1}^l -> {0, 1}^n -> {0, 1}^n
Função D: {0, 1}^l -> {0, 1}^n -> {0, 1}^n



4. **Modos de operação**:

- Os padrões de texto simples não devem ser evidentes no texto cifrado
- A eficiência do método utilizado deve ser superior à eficiência da primitiva de cifra
- A dimensão do texto cifrado deve ser aproximadamente igual à dimensão do texto simples
- A decifra deve ser capaz de detectar e corrigir erros
- Acesso aleatório - capacidade de decifrar e alterar apenas parte do texto cifrado.

Existem 2 modos de operação:

- ECB (Electronic Code Book)
- CBC (Cipher Block Chaining).

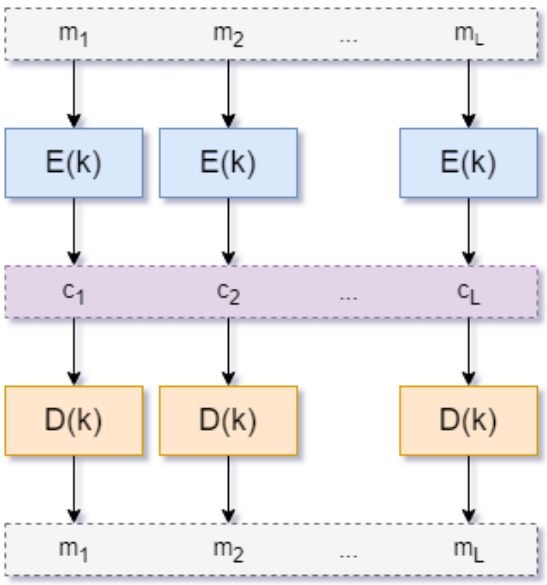
No modo **ECB**, os blocos de texto são iguais, cifrados com a mesma chave e implicam blocos de texto cifrados iguais. A cifra é executada de forma independente em cada bloco. A ocorrência de erros num bloco de texto cifrado afeta apenas a decifra desse bloco (propagação de erros) e o acesso é aleatório.
No modo **CBC**, sob a mesma chave e sob o mesmo vetor de inicialização (IV), duas mensagens iguais implicam textos cifrados iguais. A cifra é executada de forma independente em cada bloco. A ocorrência de erro num bloco de texto cifrado cj afeta a decifra desse bloco e do próximo bloco cj+1 (a decifra do bloco cj+1 terá erros nas mesmas posições do bloco cj). Reordenar os blocos de texto cifrados afeta a decifra e é relativamente fácil manipular um determinado bloco de texto simples.

NOTA: IV é um vetor de inicialização. Estes devem ser armazenados e transmitidos de forma simples e não devem ser reutilizados (exclusividade). Não deveria ser previsível (imprevisibilidade).

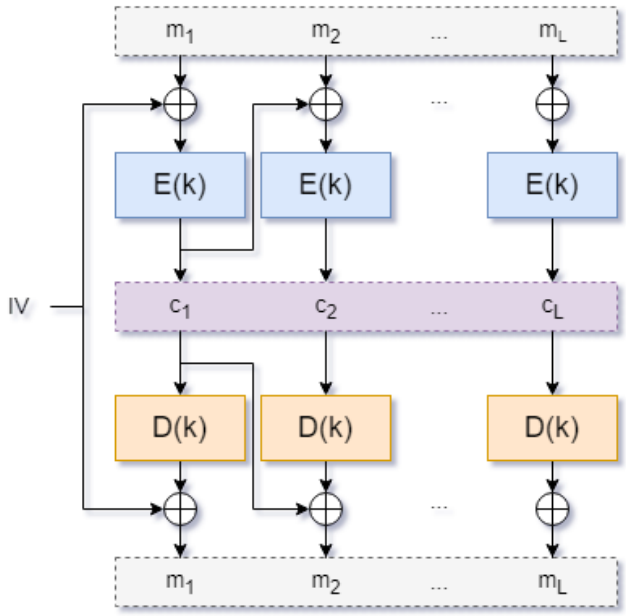
O **padding** é uma forma de pegar nos dados que podem ou não ser múltiplos do tamanho do bloco de uma cifra e estendê-los para que assim sejam.

- X é a contagem de bytes a serem adicionados à dimensão da mensagem para torná-la um múltiplo da dimensão do bloco.
- Exemplo: padding PKCS#5 e PKCS#7.

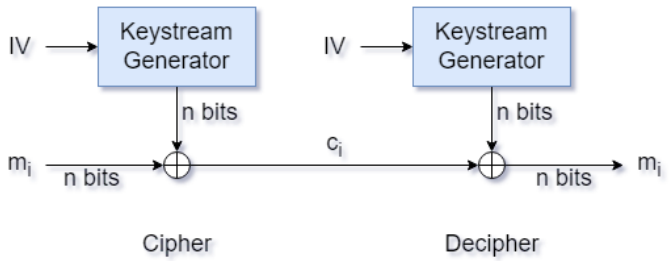
ECB (Electronic Code Book)



CBC (Cipher Block Chaining)



Modos de operação Stream: Modos como CBC precisam de algoritmos diferentes para cifrar e decifrar.



CTR (Counter) Mode

- Sob a mesma chave e sob o mesmo vetor de inicialização (IV), duas mensagens iguais implicam textos cifrados iguais
- (Propagação de erro) A ocorrência de erro num bloco de texto cifrado cj afeta apenas a decifra desse bloco. A decifra do bloco cj terá erros nas mesmas posições do bloco cj
- O acesso é aleatório
- É relativamente fácil manipular um determinado bloco de texto simples.

Cifra autenticada

- Para garantir a confidencialidade e simultaneamente a autenticidade, é necessário utilizar uma combinação dos esquemas Cipher e MAC
- Existem duas abordagens: (Encrypt-then-MAC) A tag indica se houve alteração ou não no texto cifrado e (MAC-then-Encrypt) a tag é gerada sobre a mensagem, sendo então toda criptografada;
- Existem modos de operação cujo objetivo é produzir uma cifra autenticada combinando as operações em um único algoritmo: GCM, OCB e CBC-MAC (CCM).

Funções de Hash

- H: {0, 1}^* -> {0, 1}^n;
- n – tamanho do hash;
- **Input** - sequência binária de comprimento finito;
- **Output** - sequência binária de comprimento fixo (n)

Propriedades:

- Propriedades de segurança: É computacionalmente fácil obter H(x) dado x
- É computacionalmente difícil obter x', dado x, tal que x' != x e H(x') = H(x) -> segunda pré-imagem
- É computacionalmente difícil obter (x, x'), tal que x != x' e H(x) = H(x') -> colisão
- O hash de m representa m ("assinatura digital")
- É baseado em operações booleanas e aritméticas

Exemplos:

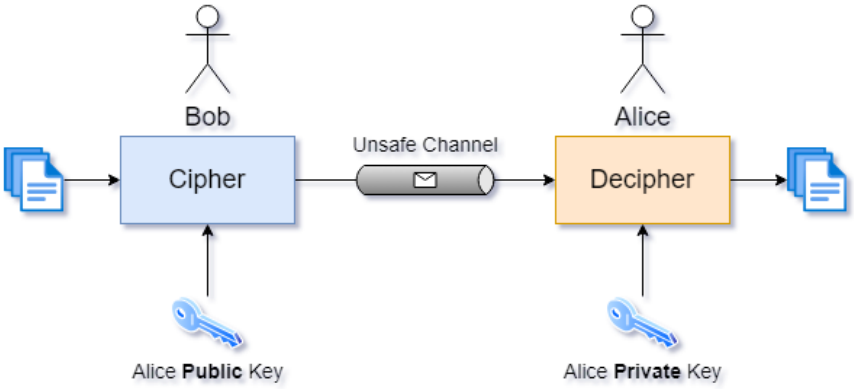
- Integridade de dados
- Derivação de chaves a partir de passwords
- Algoritmos MAC
- Assinatura digital (esquema assimétrico)
- Vários protocolos criptográficos

Funções Hash com key: É habitual designar um esquema MAC, com um algoritmo determinístico T, como uma função hash chaveada (Keyed-Hashing for Message Authentication, HMAC). HMAC é um conjunto de algoritmos MAC para usar com diferentes funções hash H.

Esquemas Assimétricos

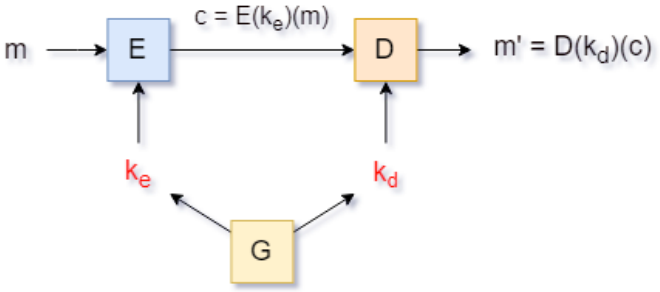
- O processo de proteção e desproteção utiliza chaves diferentes
- As chaves geralmente são usadas durante um longo período de tempo
- Existem chaves públicas e chaves privadas
- Usado em: cifra assimétrica e assinatura digital

1. Cifra assimétrica



Algoritmos (G, E, D)

G	função probabilística para geração de chaves	G :-> KeysPairs, where KeysPairs = PublicKeys x PrivateKeys
E	função probabilística para cifra	E: PublicKeys -> PlainTexts -> CipherTexts
D	função determinística para decifrar	D: PrivateKeys -> CipherTexts -> PlainTexts



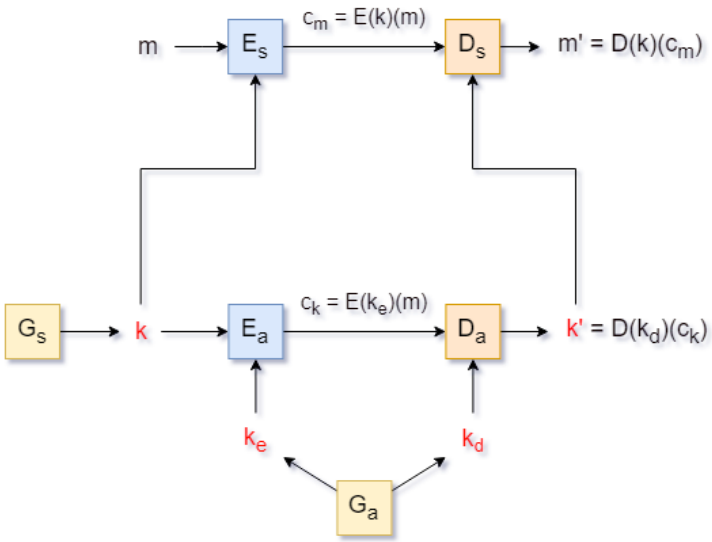
Propriedades:

- Propriedade de correção: KeyPairs: D(kd)(E(ke)(m)) = m
- Propriedade de segurança: É computacionalmente inviável obter m de c, sem o conhecimento de kd
- Esquemas assimétricos usam chaves diferentes para cifrar e decifrar
- PlainTexts: espaço de mensagens
- CipherTexts: espaço de mensagens cifradas
- Não garante a integridade dos dados
- O custo computacional é significativamente maior que os esquemas simétricos
- Limitações na dimensão da informação cifrada
- Esquemas híbridos: Esquema assimétrico usado para cifrar uma chave simétrica (transporte de chave) e esquema simétrico utilizado para cifrar as informações

2. Primitiva RSA

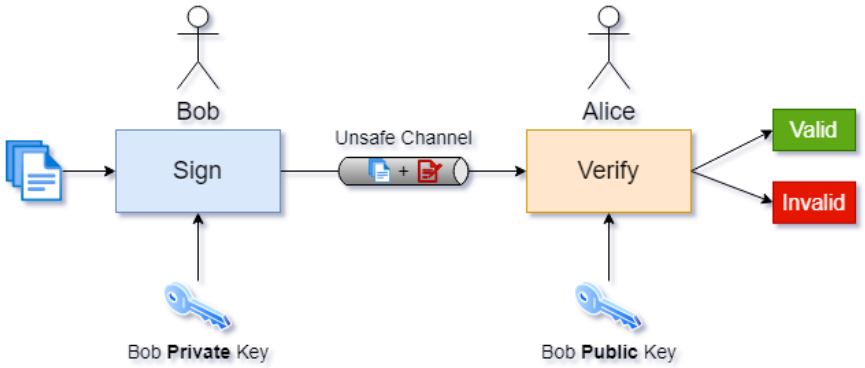
- P e Q são primos distintos e N = PQ:
  Dimensões típicas: 2^1023 <= N <= 2^4095;
- E e D tais que ED mod (P-1)(Q-1) = 1;
- Pares de chaves: Chave pública: (E, N) e Chave privada: (D, N);
- Operação pública - usada na cifra: C = M^E mod N;
- Operação privada - usada na decifração: M = C^D mod N;
- A fatorização de números primos é o problema que suporta a primitiva RSA.

3. Esquema Hibrido



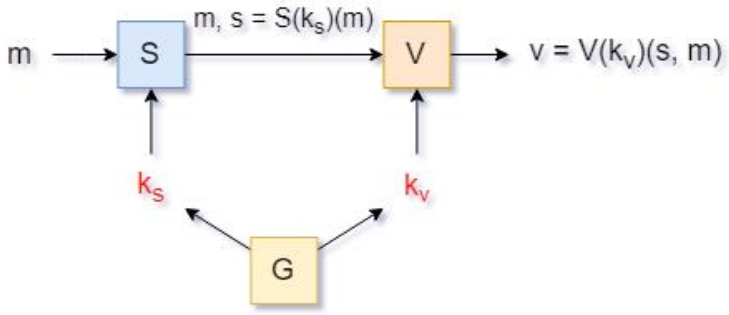
4. Assinatura Digital

- Cada participante possui 1 par de chaves para cada identidade digital
- O processo de assinatura usa chave privada
- O processo de verificação utiliza chave pública
- Os pares de chaves geralmente são usados durante um longo período de tempo
- Chave pública difundida através de certificado



Algoritmo (G, S, V):

G	função probabilística para geração de chaves	G :-> KeyPairs, where KeyPairs = PublicKeys x PrivateKeys
S	função probabilística para assinatura	S: PrivateKeys -> {0,1}* -> Signatures;
V	função determinística para verificação	V: PublicKeys -> (Signatures x {0,1}*) -> {true,false}

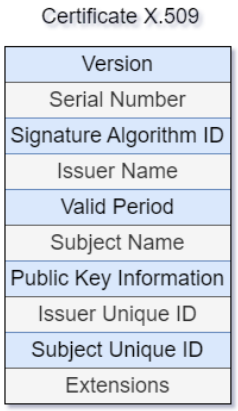


Propriedades:

- Propriedade de correção: KeyPairs: V(kv)(S(ks)(m),m) = true;
- Propriedade de segurança: Sem o conhecimento de ks é computacionalmente inviável: falsificação seletiva - dado m, encontre s tal que V(kv)(s, m) = verdadeiro; falsificação existencial - encontre o par (m, s) tal que V(kv)(s,m) = verdadeiro;
- As assinaturas normalmente têm uma dimensão fixa (ex: 160, 1024, 2048 bits);
- O custo computacional é significativamente maior que os esquemas simétricos;
- Os esquemas assimétricos utilizam chaves diferentes para assinatura e verificação; A mensagem m é uma sequência de bytes de dimensão variável; Assine!= Cifrar e Verifique!= Decifrar.

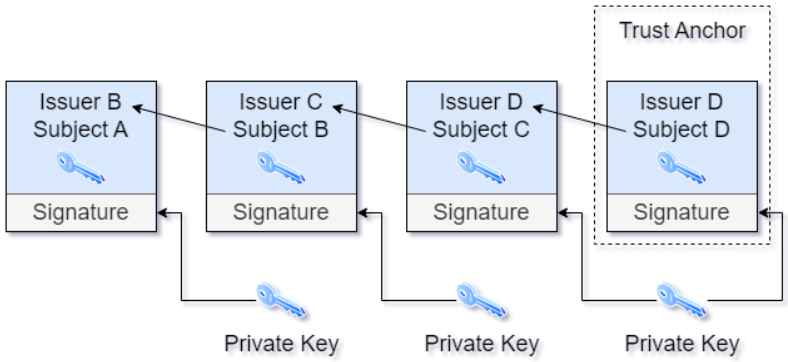
Certificados

Um certificado digital é um ficheiro ou password eletrónica que comprova a autenticidade de um dispositivo, servidor ou utilizador por meio do uso de criptografia e infraestrutura de chave pública (PKI).



Certificados X.509
Emissor: autoridade certificadora (CA)
Conteúdo: associação entre uma chave pública e um nome (identidade)
Atributos: nome, validade, extensões, ...
Assinatura do emissor: assinatura digital realizada com a chave privada da CA.

Caminho de Certificação: conjunto de certificados que permite a validação de um certificado
Recursão: Obter chave pública -> Validar Certificado -> Obter chave pública -> Validar Certificado -> ...
Condição de paragem: thrust anchor - certificado autoassinado.



Os certificados armazenam apenas a chave pública. A chave privada fica "associada" ao certificado num armazenamento seguro (por exemplo, no cartão de cidadão).

As extensões são a forma normalizada de adicionar informações não consideradas no padrão base.
Composição da extensão: Identificador de extensão, valor da extensão e sinalizador crítico (se verdadeiro, a extensão não pode ser ignorada).
Perfil: Conjunto de extensões e semânticas utilizadas para uma determinada finalidade.
Por exemplo: PKIX, S/MIME, ...

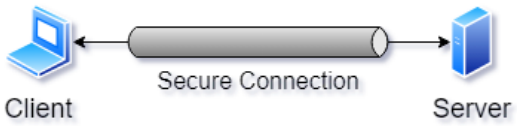
PKIX – Public Key Infrastructure for the Internet
Authority Key Identifier – identificador da chave do emissor
Subject Key Identifier – identificador da chave do assunto
Key Usage – usos permitidos para o par de chaves
Nome Alternativo – nome alternativo (e-mail, IP, URI)
Identificadores de Política – identificador de política
Restrições Básicas – restrições de uso de certificados
Restrições de nome – restrições de espaço de nome para o certificado
Restrições Políticas – restrições políticas
Uso estendido de chave – usos permitidos para o par de chaves
Pontos de Distribuição de CRL – pontos de distribuição das listas de revogação

Transport Layer Security

Transport Layer Security (TLS) é um protocolo criptográfico projetado para fornecer segurança nas comunicações em uma rede de computadores. O TLS é frequentemente usado em conjunto com o protocolo Secure Sockets Layer (SSL) para proteger as comunicações HTTP, mas também pode ser usado com outros protocolos. TLS é um padrão IETF e é especificado na RFC 5246.

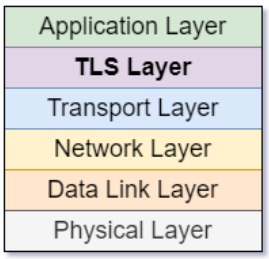
Protocolo que fornece um canal seguro entre dois endpoints
O canal tem três propriedades:

- Confidencialidade – ninguém além dos endpoints pode ver o conteúdo dos dados transmitidos
- Integridade – ninguém além dos endpoints pode alterar o conteúdo dos dados transmitidos
- Autenticação – pelo menos um endpoint do canal precisa ser autenticado para que o outro endpoint tenha garantias sobre a quem está conectado.



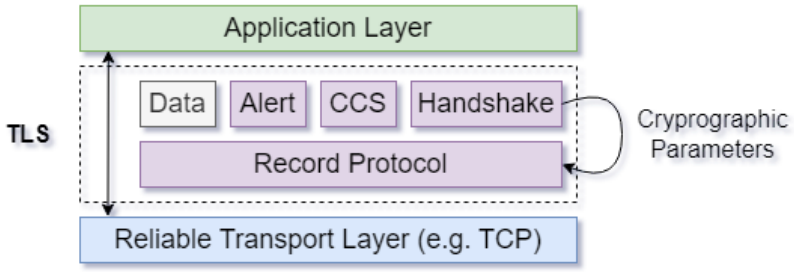
1. Camada TLS

A camada TLS é um protocolo executado entre a camada de aplicação e a camada de transporte
Os dados não criptografados são fornecidos pela camada de aplicação
O TLS criptografa e autentica os dados que envia/recebe da camada de transporte
O TLS requer uma camada de transporte confiável, como o TCP.



O TLS é dividido em dois subprotocolos:

- Protocolo Handshake – estabelece o canal seguro
- Record Protocol - transmite os dados criptografados.



2. Record Protocol

Fragmenta, compacta, autentica (MAC) e criptografa os dados
Mesma conexão TCP, com dois fluxos de dados independentes - gravação do cliente e gravação do servidor
Chaves, IVs e números de sequência são diferentes para cada fluxo
A repetição de mensagens é detectada pelo Número de Sequência
A reflexão da mensagem é detectada pelo MAC, através de chaves separadas para cada direção
Reutilização de Keystream: chaves e IVs separados para cada direção
Análise de tráfego através de chaves de criptografia separadas

3. Handshake Protocol

Responsável por:

- Negociação dos parâmetros operacionais;
- Autenticação dos endpoints;
- Estabelecimento de uma chave segura;

Autenticação de endpoint e estabelecimento de chave: A autenticação é opcional em ambas as extremidades
Suporta diversas técnicas criptográficas:

- Transporte chave – RSA;
- Acordo-chave – DH;

ClientHello	C → S: client capabilities
ServerHello	C ← S: parameter definitions
Certificate	C ← S: server certificate (**KeS**)
CertificateRequest(*)	C ← S: Trusted CAs
ServerHelloDone	C ← S: synchronization
Certificate(*)	C → S: client certificate (**KvC**)
ClientKeyExchange	C → S: **Enc(KeS: pre_master_secret)**
CertificateVerify(*)	C → S: **Sign(KsC: handshake_messages)**
ChangeCipherSpec	C → S: record protocol parameters change
Finished	C → S: **{HMAC(master_secret, handshake_messages)}**
ChangeCipherSpec	C ← S: record protocol parameters change
Finished	C ← S: **{HMAC(master_secret, handshake_messages)}**

4. **HTTPS**

- HTTP sobre TLS;
- Porta de omissão: 443;
- Verifique entre o URI e o certificado: extensão subjectAltName do tipo dNSName (se existir), o campo Nome comum no campo Assunto.

Repetição e modificação de mensagens: A modificação da mensagem de handshake é detectada com a mensagem Finished, o que garante que ambos os endpoints recebam a mesma mensagem. A repetição da mensagem de handshake implica que a mensagem Finished é diferente para cada handshake.

Master Secret: A mudança de chaves com RSA implica que o navegador utilize a chave pública do servidor para criptografar o segredo pré-mestre. O servidor descriptografa o master secret usando sua chave privada. Este processo é seguro e garante a confidencialidade do segredo pré-mestre.
O sigilo de encaminhamento perfeito é propriedade do handshake que garante que, caso a chave privada seja comprometida, não é possível descriptografar master secrets anteriores e, consequentemente, não é possível descriptografar mensagens do record protocol.

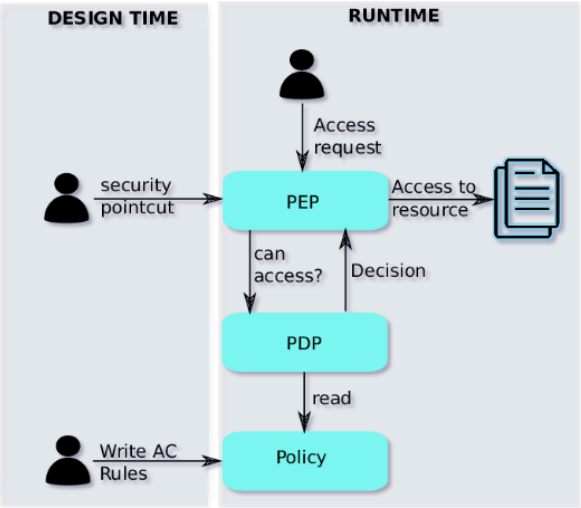
Autenticação: A autenticação é o processo de verificação de uma reivindicação de identidade.

Motivações:

- Usado no modelo de controle de acesso para determinar se um utilizador tem permissão para acessar um recurso
- Utilizado em ações de personalização
- Informações de auditoria

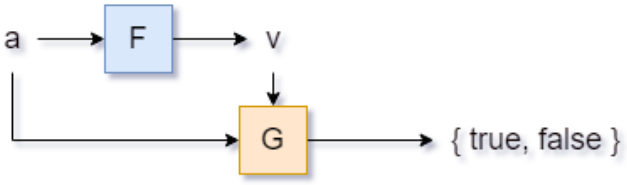
Exemplo: utilizador - identificação;
password - autenticação;

Access Control



1. Sistema de autenticação

A – conjunto de informações de autenticação
V - conjunto de informações de validação
f: A -> V - função de autenticação
g: V -> A -> {true, false} - função de validação



Exemplo: f(a) = H(a) - função hash
g(v)(a) = v == H(a) - comparação de hash;

Ataques de dicionário: Um invasor tenta adivinhar a senha tentando todas as palavras de um dicionário; Proteção contra este ataque

- Aumentar a incerteza da senha
- Controlar o acesso às informações de verificação
- Aumente o tempo de processamento da função de autenticação
- Aumente o tempo de processamento da função de validação.

2. Autenticação Web

HTTP (HyperText Transfer Protocol) é um protocolo para **transferência de dados** entre um cliente e um servidor. É stateless o que o servidoro servidor não lembra do cliente, as solicitações são independentes e sem contexto. Para manter estado, o servidor usa cookies.

Cookies HTTP: Mecanismo para armazenar e recuperar informações sobre o cliente; Mantém informações sobre o estado do cliente. São usados por meio de cabeçalhos HTTP e contém: informações sobre o estado do cliente, na forma de um par chave-valor; range de URLs para os quais o cookie é válido e data de validade, se o cookie for persistente. Pode ser usado para: criar sessões; login automático, registar histórico de navegação.

3. Processo de autenticação

Fase 1: Verifique as credenciais do utilizador (por exemplo, nome de utilizador e password); Obtenha um autenticador (por exemplo, um token)
Fase 2: Apresentação do autenticador ao servidor

Objetivos do atacante:

- Falsificação existencial: obter um autenticador válido
- Falsificação seletiva: obter um autenticador válido para um utilizador específico
- Obter a chave usada para gerar os autenticadores.

4. Autenticação via Cookies

Identificador de sessão: As informações sobre a sessão são armazenadas no servidor, o cookie contém o identificador para acessar as informações da sessão e deve ser computacionalmente inviável criar um identificador de sessão válido.

MAC: As informações sobre a sessão são armazenadas no cookie, o cookie protegido por MAC e se a confidencialidade for um requisito, o cookie deverá ser criptografado.

Logout: Invalidar o identificador da sessão e colocar o cookie em numa blacklist

Proteção: Transporte via SSL (utilização da bandeira segura) e proteção do lado do cliente (uso do sinalizador httpOnly)

Autenticação via Tokens: Token é uma string que identifica exclusivamente um utilizador, um token é gerado pelo servidor e enviado ao cliente, o cliente envia o token ao servidor em cada solicitação, o servidor valida o token e envia a resposta.

5. Autenticação HTTP

Existem dois tipos de autenticação HTTP:

- Básico: nome de utlizador e senha são enviados em texto simples;
- Digest: nome de utilizador e senha são enviados em hash;

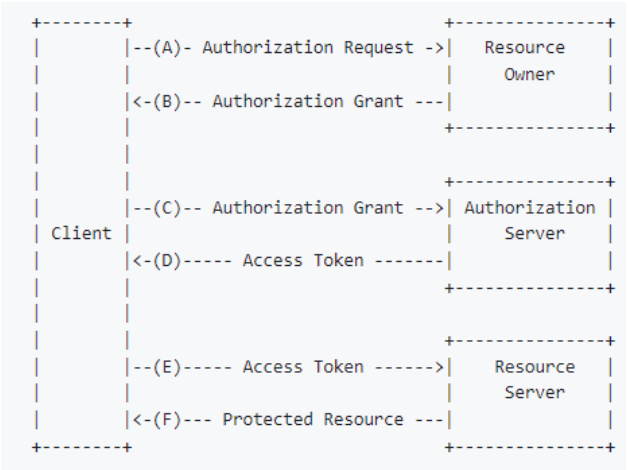
Autenticação Básica:

- O cliente acessa um recurso protegido
- O servidor responde com um código de status 401 Não autorizado e um cabeçalho WWW-Authenticate
- O cliente envia um cabeçalho de autorização com o nome de utilizador e senha em base64.

6. OAuth 2.0

Define as seguintes funções:

- Proprietário do recurso: a entidade que pode conceder acesso a um recurso protegido. Quando o proprietário do recurso é uma pessoa, ele é denominado utilizador final
- Cliente: Um aplicativo que faz solicitações de recursos protegidos em nome do proprietário do recurso e com sua autorização; quando o registro do cliente é realizado no servidor de autorização, o cliente recebe um client_id e um client_secret, que é usado para autenticar o cliente no servidor de autorização;
- Servidor de Recursos: O servidor que hospeda os recursos protegidos, capaz de aceitar e responder a solicitações de recursos protegidos usando tokens de acesso
- Servidor de Autorização: O servidor que emite tokens de acesso ao cliente após autenticar com sucesso o proprietário do recurso e obter autorização; este servidor também pode ser igual ao servidor de recursos.



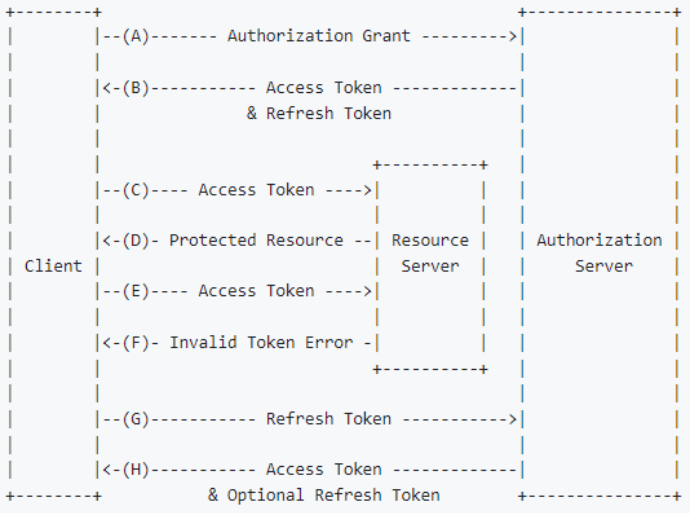
Exemplo: Um utilizador (proprietário do recurso) deseja conceder acesso a um aplicativo de terceiros (cliente) para acessar seus dados (servidor de recursos). O utilizador é redirecionado para o servidor de autorização, onde é solicitado que autentique e autorize o cliente. O servidor de autorização emite então um token de acesso ao cliente, que pode ser usado para acessar o servidor de recursos.

Access token: É uma sequência que representa as credenciais de acesso de um determinado proprietário de recurso por um determinado cliente. O cliente utiliza o access token para acessar os recursos protegidos hospedados pelo servidor de recursos em nome do proprietário do recurso. A string geralmente é opaca para o cliente.

Existem quatro maneiras (fluxos de concessão) de obter um access token:

- Credenciais do Cliente: O cliente solicita um token de acesso usando apenas suas credenciais de cliente
- Credenciais de senha do proprietário do recurso: O cliente solicita um token de acesso usando suas credenciais e a senha do utilizador; usado apenas quando o cliente é confiável
- Código de Autorização: O cliente solicita um token de acesso utilizando um código de autorização, que o proprietário do recurso obteve anteriormente
- Implícito: O código de autorização é retornado ao cliente sem uma etapa extra de troca do código de autorização.

Refresh token: um token usado para obter um novo token de acesso quando o access token atual se torna inválido ou expira, ou para obter tokens de acesso adicionais com escopo idêntico ou mais restrito (os tokens de acesso podem ter uma vida útil mais curta e menos permissões do que as autorizadas pelo proprietário do recurso).



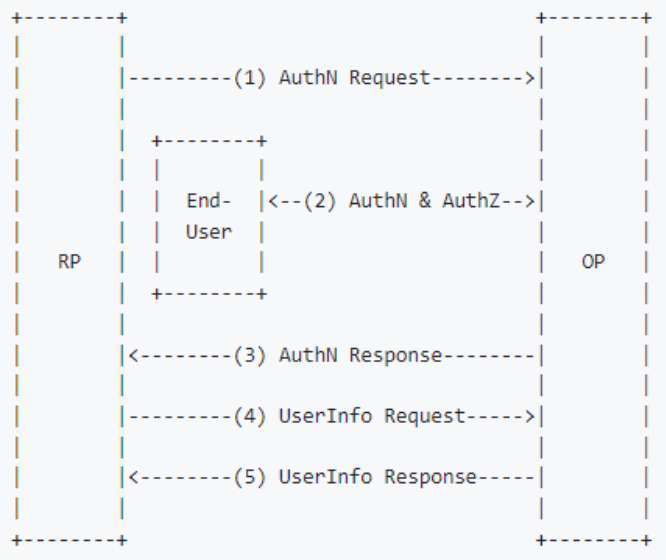
7. Front Channel e Back Channel

Front Channel: Canal de comunicação entre o cliente e o servidor de autorização, via redirecionamento user-agente. Em caso de erro, o servidor de autorização redireciona o agente do utilizador para o URI de redirecionamento do cliente com código e descrição do erro. O client_secret nunca passa pelo canal frontal.

Back Channel: Canal de comunicação entre o cliente e o servidor de autorização. A autenticação básica HTTP é usada para autenticar o cliente: client_id (nome de utilizador) e client_secret (senha);

8. OpenID Connect

End-User (utilizador): pessoa que está tentando acessar um recurso na aplicação cliente, que requer autenticação
Relying Party - RP (cliente): a aplicação cliente que está tentando acessar um recurso no servidor de recursos
OpenID Provider - OP (servidor de autorização): o servidor que armazena as credenciais do utilizador e o autentica.



Passos:

1. O RP envia uma solicitação de autenticação ao OP
2. O OP autentica o utilizador e obtém autorização
3. O OP envia uma resposta de autenticação ao RP
4. O RP envia uma solicitação ao OP para obter informações do utilizador
5. O OP envia uma resposta ao RP com as informações do utilizador

UserInfo Endpoint: Contém as informações de um utilizador autenticado. O RP envia uma solicitação ao OP para obter informações do utilizador. A resposta é um objeto JSON com as informações do utilizador.