Esquemas Simétricos

- Mesma chave para cifra e decifra; chaves usadas pouco tempo e c/ pouca dimensão; não garante integridade, garante confidencialidade; O MAC garante autenticidade. Não é possivel obter m a partir de c, sem conhecer k. Dimensão do bloco deve ser elevada para impedir ataques
- Primitivas: AES, DES – deterministicas, com o mesmo k e o mesmo bloco m dá o mesmo resultado (usam a mesma chave)

Modo de operação ECB: Neste modo, cada bloco de texto simples é criptografado de forma independente com a mesma chave. Isso significa que o mesmo bloco de texto simples sempre será criptografado para o mesmo bloco de texto cifrado. No entanto, isso pode revelar padrões no texto cifrado se o texto simples tiver blocos repetidos (blocos de texto em claro iguais + cifrados com a mesma chave = blocos de texto cifrado iguais.). Pode usar padding

Modo de operação CBC: Neste modo, cada bloco de texto simples é combinado com o bloco de texto cifrado anterior através de uma operação XOR antes de ser criptografado. Isso garante que mesmo que o texto simples tenha blocos idênticos, os blocos de texto cifrado correspondentes serão diferentes. Além disso, o CBC usa um vetor de inicialização (IV) para o primeiro bloco para garantir a aleatoriedade. **(**sob a mesma chave e sob o mesmo IV, duas mensagens iguais implicam criptogramas iguais)

- IV (vetor de inicialização): não deve ser previsivel, deve ser armazenado e transmitido de forma simples e não deve ser reutilizado
- Padding: usado para garantir que os dados a serem criptografados sejam de um tamanho que seja múltiplo do tamanho do bloco. O algoritmo de padding consiste em colocar o valor total dos bytes em falta em cada espaço. **Limitações**: **ataques de padding** (alguns algoritmos podem ser vulneráveis a ataques onde um invasor pode decifrar os dados sem a chave), **complexidade** (a necessidade de implementar e gerenciar algoritmos de padding adiciona complexidade ao sistema de criptografia), **dados binários** (alguns esquemas de padding podem ter problemas com dados que incluem bytes de valor zero, pois esses bytes podem ser interpretados como padding) e **remoção de padding** (após a descriptografia, o padding deve ser removido. Se não for removido corretamente, pode levar a erros na interpretação dos dados)

Modo de operação em stream: não precisa de padding; sob a mesma chave e sob o mesmo IV, duas mensagens iguais implicam criptogramas iguais

MAC: mesma chave para geração de marcas (T) e verificação de marcas (V) e marca t (tag) tem tipicamente dimensão fixa. É mais rápido. Ambos os lados precisam da chave privada (esta pode ser comprometida). Usado para verificar a integridade dos dados e autenticar a mensagem, garantindo que ela não foi alterada durante a transmissão. No entanto, o MAC não fornece confidencialidade. O que significa que, embora a integridade dos dados seja protegida, o conteúdo da mensagem ainda é visível e pode ser lido por qualquer pessoa que tenha acesso à mensagem. Para garantir a confidencialidade, os dados precisam de ser criptografados antes da transmissão. Portanto, é comum usar o MAC em conjunto com a criptografia (cifra) para fornecer tanto a integridade dos dados quanto a confidencialidade (cifra autenticada)
- **Encrypt-then-MAC**: A marca indica se houve alteração ou não do criptograma - E(k1)(M) || T(k2)(E(k1)(M))
- **MAC-then-encrypt**: A marca é gerada sobre a mensagem, e é posteriormente tudo cifrado - E(k1)(M || T(k2)(M))
- Modos de operação c/ objetivo de produzir uma cifra autenticada, combinando operações num só algoritmo (GCM, OCB, CCM)

Salt: O que é? O salt é uma cadeia de caracteres aleatória usada em criptografia[1]. Ele é concatenado com uma senha antes de aplicar uma função de hash, o que resulta em hashes únicos para cada senha, mesmo que duas senhas sejam iguais. Propósito: é usado para forçar a unicidade do hash gerado, aumentando a segurança das senhas armazenadas. O que acontece se eu aumentar o tamanho do salt? O tamanho do salt determina o número de possíveis salts que podem ser usados. Se você aumentar o tamanho do salt de 16 bits para 64 bits, o número de possíveis salts aumenta de 2^16 (65,536) para 2^64. Isso significa que um atacante teria que tentar muitas mais possibilidades para adivinhar o salt, tornando o ataque muito mais difícil.

Esquemas Assimétricos

- chaves diferentes para cifra e decifra; chaves usadas durante muito tempo; não garante integridade. É impossivel obter m a partir de c, sem saber a private key. A arquitetura dos algoritmos de cifra e decifra é constituida por uma primitiva de cifra assimétrica (RSA) e um método de formatação ou padding (+ função de hash); a mesma primitiva pode ser usada com vários tipos de formatação e funções hash; a mesma chave é utilizada na geração da marca e na verificação da marca (as chaves são usadas apenas pela primitiva). "Todos podem cifrar, apenas o receptor autorizado pode decifrar" e "Todos podem verificar, apenas o emissor autorizado pode assinar (gerar a marca)"

Assinatura Digital: É mais seguro, fornece não repúdio. Cada interveniente tem 1 par de chaves por cada identidade digital. Processo de assinatura usa chave privada, processo de verificação usa chave pública e par de chaves usadas durante muito tempo

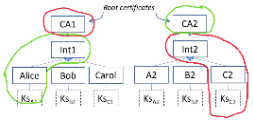
Certificados

Certificados X.509: quem certifica (emissor – autoridade de certificação AC), o que certifica (associação entre uma chave pública e um nome (identidade)), outros atributos (validade, usos da chaves, extensões), assinatura do emissor (assinatura digital realizada com a chave de assinatura (privada) do emissor). Certificados guardam a chave pública

Verificação da cadeia de certificação: Para verificar a assinatura de um certificado é preciso a chave pública do seu emissor. A chave privada é usada para assinar. A chave privada do certificado raíz assina o seu próprio certificado e o certificado do sujeito anterior

Caminho de certificação

Recursão: obter private key -> validar cert -> obter public key
Paragem: thrust anchor (certificado autoassinado)
Autenticação do servidor sem autenticação do cliente: configurar chave privada do servidor, configurar o certificado concantenado com o intermédio e adiciona-se a raíz de confiança (para o client confiar)



Autenticação do servidor sem autenticação do cliente: faz-se o mesmo processo do lado esquerdo como mostra a imagem

PKIX: Authority Key Identifier (identificador da chave do emissor), **Subject Key Identifier (**identificador da chave do subject), **Key Usage (**usos permitidos para o par de chaves), **Basic Constraints (**restrições ao uso do certificado (indica se o subject é uma autoridade de certificação e qual a maior dimensão do caminho de certificação)).
KeyStore: Armazenamento de chaves e certificados. Três tipos de entrada: Chaves privadas e certificados associados (incluindo a cadeia), chaves simétricas, certificados representando trustanchors. Protecção baseada em passwords (Integridade de todo o repositório -uma passwordpor repositório e confidencialidade das entradas contendo chaves privadas ou secretas -uma password por entrada do repositório)

Protocolo SSL/TLS

É um protocolo que fornece um canal seguro entre dois endpoints. O canal seguro tem três propriedades: confidencialidade, integridade e autenticação.

Record protocol: Fragmenta, comprime, autentica (MAC) e depois cifra. Chaves, IVs e número de sequência diferentes (client write e server write). Chaves não derivam da chave privada do servidor.
- Repetições de mensagens: vê-se pelo número de sequência; Reflexão da mensagem: chaves MAC separadas para cada direção; Reutilização de keystream (criptografia simétrica baseada em streams): chaves de criptografia e IVs separados para cada direção; Análise de tráfego: chaves de criptografia separadas

Handshake protocol: Lida com a criação e gestão de conexão segura, ou seja, o estabelecimento seguro dos parâmetros criptográficos do record protocol. É responsável por: negociação dos parâmetros de operação, autenticação dos endpoints, estabelecimento de chave segura. No handshake do TLS, cliente e servidor enviam os números client_random e server_random, sobre um canal inseguro. Se os números forem modificados no canal, tal será detetado nas mensagens finais do handshake.
O protocolo TLS usa números aleatórios gerados tanto pelo cliente quanto pelo servidor durante o processo de handshake. Esses números são únicos para cada sessão e são usados para calcular a chave de sessão. Se um atacante tentar reutilizar as mensagens de um handshake anterior, a tentativa falhará porque os números aleatórios serão diferentes e a chave de sessão calculada não será válida.
- **Ataque Man-in-the-Middle (MITM)**: Neste ataque, um invasor coloca-se entre o cliente e o servidor durante o handshake, interceptando e possivelmente alterando as mensagens trocadas. O objetivo é obter acesso não autorizado aos dados transmitidos ou modificar os dados durante a transmissão. Detetado através de autenticação mútua com certif. Digitais. O atacante inicia o processo de autenticação com o fornecedor de identidade (por exemplo, Google) e interrompe o processo quando a resposta passa pelo seu navegador.
- **Ataque de Downgrade**: Neste ataque, um invasor tenta fazer com que o cliente e o servidor concordem em usar uma versão mais antiga e menos segura do protocolo TLS ou um algoritmo de criptografia mais fraco. Isso pode permitir que o invasor decifre ou modifique os dados transmitidos. Detetado por verificação da integridade das mensagens trocadas.
- **Cross-Site Scripting (XSS)**: Scripts que simulam funcionalidades do site original. Utilizado para roubar dados de login ou elaborar outros ataques. Um cliente de e-mail pode ser alvo de um ataque de Cross-Site Scripting (XSS). Isso pode ocorrer se o cliente de e-mail permitir a execução de scripts contidos em e-mails
- ataque às autoridades de certificação:
Cipher suite define: A função de hash usada pelo HMAC (e.g. SHA), o esquema simétrico (suporta modos de bloco ou stream), esquema de estabelecimento de chaves (RSA or DH)

Quando é usado RSA para transporte de chave:
C <-> S: negociação dos algoritmos a serem usados
C <- S: certificado de servidor
C -> S: segredo aleatório cifrado com a chave pública do servidor
C <- S: prova de posse do segredo aleatório
Se for necessário autenticação de cliente:
C <- S: O servidor solicita o certificado de cliente

C -> S: certificado de cliente
C -> S: prova de posse da chave privada, assinando as mensagens anteriores

Repetição e modificação de mensagens: 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 pre master secret. O servidor descriptografa o master secret usando sua chave privada. Este processo é seguro e garante a confidencialidade do segredo pré-mestre. 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.

É insegura a troca do **pre-master secret** usando chaves públicas e privadas devido à vulnerabilidade a ataques. Especificamente, este método é suscetível a ataques de "man-in-the-middle" onde um invasor pode interceptar a chave pública e substituí-la pela sua própria. Isso permitiria ao invasor descriptografar o pre-master secret e, consequentemente, comprometer a segurança da sessão.

Perfect forward secrecy: é a propriedade do handshake que garante que, se a chave privada for comprometida, não é possível decifrar master secret anteriores (e consequemente não é possível decifrar mensagens do record protocol)
- A troca de chaves com RSA implica que o browser usa a chave pública do servidor para cifrar o pre master secret. O servidor decifra o pre master secret usando a chave privada. O que acontece se a chave privada for comprometida? O pre master secret dos handshakes seguintes e dos anteriores (guardados pelo atacante) podem ser decifrados
- Este processo é seguro e garante confidencialidade do pre master secret

Ataques a autoridades de certificação: A validação do certificado do servidor depende de uma raiz de confiança; as raízes de confiança são certificados emitidos por autoridades de certificação

Autenticação

Ataques: ataque à interface de autenticação, ataque à informação de validação.
Ataques de dicionário(não implica utilização da interface de autenticação): 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.
Protocolo HTTP: é stateless, os pedidos são independentes e sem relação (mesmo pedidos consecutivos do mesmo cliente na mesma ligação TCP)

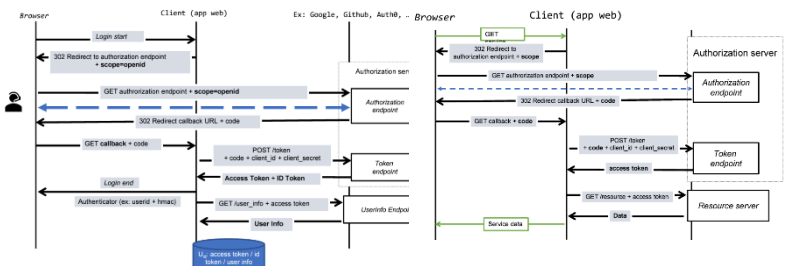
Cookies: Mecanismo para armazenar e recuperar informações sobre o cliente. Contém: informações sobre o estado do cliente; range de URLs para os quais o cookie é válido e data de validade, se o cookie for persistente. A aplicação web valida o acesso com base na autenticidade dos cookies. A falsificação de um pedido que mude o estado da aplicação pode ser feita usando pedidos cross-site (problema conhecido como Cross-Site Request Forgery). Para evitar este problema, a opção SameSite=Strict garante que cookies marcados com esta opção não são usados em pedidos de cross-site. Isso significa que, se um utilizador estiver no site siteA.com e tentar fazer um pedido para siteB.com, nenhum cookie de siteB.com será enviado com o pedido. Isso pode ajudar a prevenir ataques CSRF, onde um atacante tenta enganar um utilizador para que faça um pedido indesejado para um site onde o utilizador está autenticado.

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.

OpenID Connect e OAuth 2.0

id_token: conjunto de asserções sobre um utilizador autenticado
client_id e client_secret são o username e password, respetiv. Os clientes têm de se registar no servidor de autorização, onde lhes é atruido um client_id e um client_secret, usado pelo cliente no servidor de autorização.



access_token: representa uma credencial de acesso. String opaca para os clientes, representando scopes específicos e duração de acesso. É emitido pelo servidor de autorização e validado pelo servidor de recursos. Tem a vantagem do servidor de recursos não precisar de saber lidar com um conjunto diversificado de mecanismos de autenticação. Formato, estrutura e métodos de utilização podem variar. Os clientes acedem aos recursos protegidos indicando um access token, o servidor de recursos usa-o para aplicar políticas de acesso
refresh_token: usado para pedir novas credenciais de acesso. O AuthorizationServer entrega também um refreshtoken usado para pedir novas credenciais de acesso
scopes: representam o tipo de autorização que está a ser pedido a determinado recurso. Cada scope é uma string. Cada pedido de autorização contém zero ou mais scopes.
authorization code grant: código de autorização é obtido pelo dono dos recursos e entregue ao cliente para que este obtenha o access_token. A passworddo dono dos recursos não é visível para o cliente. Não define forma do dono de recursos se autenticar e dar consenso para o cliente aceder ao recurso. O **parametro state** é usado para mitigar ataques de falsificação de solicitação entre sites (CSRF). O cliente gera um valor único para state e o envia na solicitação de autorização. Quando o servidos de autorização responde com o código de autorização, ele inclui também o mesmo valor state inalterado na resposta. Apenas há redirect em comunição com o authorization endpoint, e que com o resource server é diretamente
O cliente então compara o valor state na resposta com o valor state que ele enviou originalmente na solicitação. Se os valores corresponderem, o cliente sabe que a resposta está relacionada ao pedido de autorização original. Isso ajuda a garantir que a resposta não foi adulterada e é válida

client credentials grant flow: Autorização é dada somente com base nas credenciais do cliente (client_ide client_secret), ou seja, não está envolvido nenhum utilizado
resource owner password credential grant: Autorização é dada com base na passworddo utilizador. A aplicação cliente não precisa de persistir a password. Deve apenas ser usado quando há um nível elevado de confiança entre o dono de recursos e a aplicação cliente (ex: cliente faz parte do sistema operativo)
front channel: Termo usado para designar o canal de comunicação client<-> Authorization end point, via redirecção do user-agent. Em caso de erro a resposta tem sempre de ser entregue via redirect (não podem ser usados os códigos de erro). O client_secret nunca passa pelo frontchannel. Usando o frontchannel como é que o cliente estabelece uma relação entre pedido e resposta? Através do parâmetro state
back channel: Termo usado para designar o canal de comunicação cliente <-> Token endpoint. Usa mensagem POST HTTP e os respectivos códigos de erro. HTTP Basic authentication com username(client_id) e password (client_secret)

RBAC

- $U = \{u_1, u_2, u_3, u_4\}$
- $RH = \{M \preceq T, M \preceq D, D \preceq S, T \preceq S, T \preceq T_2, D \preceq D_2\}$
- $UA = \{(u_1, M), (u_2, T_2), (u_3, D_2), (u_4, S)\}$
- $PA = \{(M, p1), (D, p2), (T, p3), (D_2, p5), (T_2, p4)\}$

O u4 tem o papel de Supervisor (S). De acordo com a hierarquia de papéis definida em RH, um Supervisor tem acesso a todos os outros papéis (Membro, Desenvolvedor e Testador). Portanto, o conjunto total de permissões que podem existir numa sessão com o usuário u4 seria a união de todas as permissões associadas a esses papéis, ou seja, {p1, p2, p3, p4, p5}

U – conjunto de utilizadores; **RH** – relação de hierarquia de funções(roles) - Role Hierarchy (T herda tudo de M); **UA** – relação entre o utilizador e a sua função – User to Role Assignment (ex: u1 tem a função de Membro); **PA** – relação de atribuição de permissão para função(roles) – Permission to Role Assignment
Conceito de sessão: roles que o utilizador está a exercer (pode mudar de acordo com as necessidades do momento)

(1,5) No sistema Moodle os professores podem assumir o papel de "professor", "professor não editor", "aluno" ou "visitante" mas, para cada professor, apenas um destes papéis pode estar ativo em cada interação com o sistema. Assuma que os papéis não estão relacionados entre si. Explique sucintamente como o RBAC pode, em geral, ser usado para especificar esta política.

Roles: professor, professor não editor, aluno, visitante
UA: = {(Prof1, professor), (Prof1, professor não editor), (Prof1, aluno), (Prof1, visitante)}.
Sessões: Quando um professor interage com o sistema, ele inicia uma sessão. Nessa sessão, ele pode ativar apenas uma das funções atribuídas a ele. A função ativa determina as permissões que o professor tem durante essa sessão.
Permissões: As permissões seriam definidas para cada função. Por exemplo, a função "professor" pode ter permissões para criar e editar cursos, enquanto a função "professor não editor" pode ter permissões para visualizar, mas não editar cursos.

Classe Cipher JCA: ENCRYPT_MODE: Criptografia de dados, DECRYPT_MODE: Descriptografia de dados, WRAP_MODE: Agrupar um java.security.Key em bytes para que a chave possa ser transportada com segurança, UNWRAP_MODE: Desempacotamento de uma chave previamente empacotada em um objeto java.security.Key. Para criptografar ou descriptografar dados em uma única etapa, chame um dos métodos doFinal, o update é para múltiplos e Uma chamada para doFinal redefine o objeto Cipher para o estado em que estava quando inicializado por meio de uma chamada para init. Ou seja, o objeto Cipher é redefinido e fica disponível para criptografar ou descriptografar
Padding PKCS#7: Se 5 bytes de padding são necessários, então os 5 bytes finais do bloco de dados serão todos 05. O PKCS#5 é definido para tamanhos de bloco de 8 bytes, enquanto o PKCS#7 funcionaria para qualquer tamanho de bloco de 1 a 255 bytes. PKCS#5 não dava para ser usado com AES pq ele usa 16 bytes
Casbin: ficheiros .conf (Define a estrutura das políticas de controle de acesso) .csv (Contém as políticas de acesso reais que são aplicadas com base no modelo definido em file.conf)
JCA – tem bibliotecas para implementação de algoritmos criptográficos