1. **O que é um sistema distribuído?**

R = Sistema distribuído pode ser definido como um conjunto de computadores independentes (podem pertencer a diferentes organizações) que para os usuários parecem ser um sistema único e harmonioso.

2. **O que é middeware?**

R= É o software que conecta programas que são separados e já existentes, geralmente ele se encontra entre a app e o sistema operacionais.

3. **Quais são os objetivos de um sistema distribuído?**

R= Fazer a ligação entre usuários e recursos (como compartilhamento de recursos), transparência (como iludir para o usuário pensar que o sistema é único, oculta falhas...), escalabilidade (medir a capacidade do sistema em lidar facilmente com uma quantidade crescente de trabalho(aumento de carga, número de utilizadores, entre outras..)).

4. **Um estilo arquitetónico é determinado a partir dos:**

R= Componentes (modulo com interfaces requeridas e fornecidas), conexões (modo como os componentes estão ligados), dados intercambiados (troca de dados entre componentes), formas de configuração (configuração dos componentes).

5. **O que é a arquitetura cliente-servidor?**

R= É uma estrutura que distribui as tarefas e cargas de trabalho entre o requente (cliente) e os fornecedores (servidor). O ruim é que se o servidor cai, as solicitações do cliente podem não ser satisfeitas. Um ponto positivo é que como a informação esta centralizada em um só lugar acaba por estar mais segura.

6. **O que é arquitetura P2P?**

R= Se conectam vários nós de maneira aleatória que compartilham entre si vários tipos de dados, cada um pode tanto fazer o papel de servidor enviando dados, ou de cliente requisitando dados.

7. **Qual a diferença entre o cliente servidor e o P2P?**

R= No P2P todos os nós tem os mesmos benefícios, então se um nó falha os outros continuam ativos continuando o trabalho. Já o cliente-servidor se o servidor cai, as solicitações do cliente podem não ser satisfeitas, no entanto um ponto positivo é que como a informação esta centralizada em um só lugar acaba por estar mais segura.

8. O que é código móvel?

R= O pedido do cliente desencadeia o download de um objecto executável que é enviado para o cliente e executado localmente (Ex: Applet Java), o que melhora o desempenho e interactividade

9. O que é TO Marshall?

R= Converter os dados da minha representação local para a representação na rede

10. O que é To unmarshall?

R= É o que faço da receção, leio os dados das mensagens e converto do formato da mensagem para no formato de representação dos meus dados.

11. Corban Common Data Representation

R= Os dados são transmitidos no formato do emissor, que é especificado em cada mensagem, o receptor converte se tiver uma convenção diferente**.**

12. O que é RPC?

R= É permitir que sejam estruturados o cliente e o servidor exatamente da mesma forma que os programas funcionam, com chamada a funções. Ele se encaixa perfeitamente no modelo cliente-servidor.

- ***<u>Benefícios</u>*:**
  - Adequa-se ao fluxo de execução das aplicações
  - Simplifica tarefas fastidiosas e delicadas (heterogeneidade convertendo o formato dos dados locais para formato da rede.)
  - Simplifica a divulgação de serviços (servidores)
  - Esconde diversos aspetos específicos do transporte (Tratamento de erros, Endereçamento do servidor).

13. Invocação de um RPC

O cliente chama o stub do cliente que vai converter os parâmetros do formato local para o formato rede e formatar uma mensagem que a seguir chama o Runtime do RPC para enviar a mensagem ao servidor que é recebida também pelo Runtime RPC agora do lado do servidor, e depois vai chama a rotina Stub servidor que vai ler os parâmetros da mensagem para variáveis locais, vai converte-los no formato de rede para formato local e a seguida chama a função no servidor e depois fazer todo o caminho de volta até chegar ao cliente.

14. O que preciso para fazer um sistema de RPC?

R= Em termo de desenvolvimento o sistema de RPC precisa de uma Interface Description Language (IDL) esta linguagem serve para que o servidor descreva qual a interface que exporta e para que os clientes possam também entender. E um compilador de IDL que dadas as interfaces descritas das interfaces gerem os stubs. Em tempo de execução deve ter o serviço de nomes para que o servidor se registe e os clientes o possam encontrar e deve ter biblioteca de suporte para fazer o registo do servidor.

15. Quais são as operações/rotinas que posso executar do lado do servidor (condiciona a semântica da invocação)?

R= **Operações idempotentes** = operação que pode ser realizada várias vezes, isso significa que o efeito final sobre o sistema de executar mais de uma vez é o mesmo de executar apenas uma vez. Ex: leitura de valores

Operações não idempotentes = só podem ser realizadas uma vez. Ex: acumulo de valores.

16. Assume que as máquinas são fail-silent:

- **Talvez (may-be)** – não sei a rotina foi chamada ou não remotamente, porque não sei se foi o pedido que se perdeu, ou o servidor falhou, ou a mensagem de resposta falhou... só sei que tive um erro.
- **Pelo-menos-uma-vez (at-least-once) -** se eu receber um resultado da chamada da rotina então sei que foi executada pelo menos uma vez.
- **No-máximo-uma-vez (at-most-once) – s**e usar um protocolo com ligação do tipo TCP (dependente de confirmação de envio e receção de pacote) para enviar mensagem eu sei que se eu recebi um erro a rotina foi executada no máximo uma vez.
- **Exactamente-uma-vez (exactly-once) –** se não recebi erro sei que foi executa exatamente uma vez.

17. A semântica pode ser impostas pelo mecanismo de transporte:

- Sun RPC sobre **UDP**/IP:
  - **Se houver resposta**: pelo-menos-uma-vez, porque tenho resposta então quer dizer que foi executada, mas pode ser que deu timeout e eu repeti o pedido.
  - **Sem resposta**: talvez, porque não sei se a rotina foi executada no servidor.

- Sun RPC sobre TCP/IP:
  - Se houver resposta: exactamente-uma-vez, sei que não a pedidos duplicados no servidor então se eu tiver respostas, sei que a rotina foi executada exatamente uma vez.

- Sem resposta: no-máximo-uma-vez, não sei se foi o pedido que não chegou, ou se a resposta não chegou ou se o servidor crashou depois de ter executado a função, então que se executou foi no máximo uma vez porque ele não duplica pedidos.

18. RPC – representação dos dados:

- ***Estrutura dos dados***
  - ♣ Implícita
  - ♣ Autodescritiva (marcada, tagged)
- Políticas de conversão dos dados
  - ♣ Canónica, ou seja há protocolo de representação na rede e o rpc converte sempre os dados para esse protocolo de representação standard.
    - N formatos ⇒ N funções
    - Não há comportamentos variáveis
    - É preciso converter mesmo quando é inútil
  - ♣ O-receptor-converte (Receiver-makes-it-right)
    - Poupa conversões inúteis
    - N formatos ⇒ N x N funções (torna a escalabilidade e a extensibilidade mais difíceis).

19. Execução de RPCs:

► Um pedido de cada vez

♣ Serialização de pedidos

♣ Uma única thread para todos os pedidos

► Vários pedidos em paralelo

♣ Uma thread por pedido

♣ A biblioteca de RPC tem que suportar paralelismo:

• Sincronização no acesso a binding handles

• Sincronização no acesso a canais de comunicação

Obs: estabelecem um numero máximo que podem criar a medida que elas são necessárias, porque se o ritmo de chegada dos pedidos é superior a velocidade que eu posso processar os pedidos, então crio mais threads e vai consumindo os recursos da maquina e nenhuma avança e mais tarde isso vai fazer o servidor entrar em colapso.

20. Chamadas em ricochete (callbacks)

É quando o servidor esta a executar o pedido mas precisa devolver o controle ao cliente porque o cliente precisa adicionar alguns dados que não estava preparado para enviar logo no pedido original. Isso não é boa ideia, primeiro porque o cliente tem que

se comportar também como servidor, deixa de ser simples aquele que só faz pedidos e recebe respostas para ter a capacidade de receber algo que não é uma resposta e conseguir processa-la. É também ruim para o servidor.

21. O que são os stubs?

R= Os stubs são códigos que convertem os dados do lado do cliente depois chamam a rotina de RPC para enviar a mensagem, recebem a resposta depois os convertem.

22. O que é web service?

R= O objetivo dos web services é permitir que um servidor de uma organização disponibilize serviços na forma de rotinas que podem ser chamadas nesse servidor, porém serviços esses que podem ser chamados por qualquer cliente, qualquer organização…Portanto tem que definir uma linguagem de definição de interfaces que tem que ser completamente independente das linguagens de programação.

- Web Services - maior flexibilidade e interoperabilidade
  - Definição das interfaces de forma independente das linguagens de programação (WSDL)
  - Clientes e servidores podem estar escritos em linguagens diferentes.
  - Formato das mensagens independente da implementação (SOAP)- completamente standard.

23. Diferença entre o RPC e JAVA RMI do SOAP

Tanto no RPC quanto no JAVA RMI é que nem um sistema nem outro descrevia qual era o tipo da mensagem que circulava na rede, era a própria biblioteca que implementa esse formato, mas não é publico. Já no web service como quero que qualquer cliente possa me enviar mensagem deve ter o formato publico e nesse exemplo é o SOAP.

24. JAVA RMI ou WEB service com SOAP?

Para a minha empresa eu optaria em utilizar o web service com SOAP, porque ao utilizar o JAVA RMI teria de garantir que toda a gente está a utilizar o java e ainda assim que seja a mesma versão do java para garantir que o formato da mensagem seja exatamente o mesmo. Ou seja, todos os participantes teriam de usar elementos de tecnologias sincronizadas umas com as outras o que cria uma dependência muito grande.

25. Web services: arquitetura:

- **UDDI:** descobrir quais são os webs services existentes.
- **WSDL:** Integra a linguagem de definição das interfaces.
- **SOAP:** forma de utilização do XML para formatar as mensagens
- **HTTP, JMS, SMTP:** Protocolo de comunicação

26. Componente da WSDL:

Permite definir qual é a interface e ao mesmo tempo também saber quais são os end points de acesso, que é o url onde essa interface esta definida.

27. Elementos do WSDL:

•**Service:** definição da localização (URL) do serviço (faz associação do portType com onde ele está localizado)

• **Port:** definição do endereço físico (URL) do serviço (é para dizer o endereço em que posso enviar os pedidos).

• **Binding:** definição do formato da mensagem (associar qual o protocolo de representação)

• **PortType**: definição dos métodos a invocar (é o que define qual interface posso chamar remotamente, de certa forma equivale à interface do java).

• **Message:** definição das mensagens trocadas (conjunto de partes que cada uma descreve os argumentos e os respetivos tipos.)

• **Types:** definição dos tipos de dados utilizados (definição abstrata dos tipos definidos pelo utilizador)

28. Sobre o SOAP:

- Ele especifica o formato das mensagens
- Uma mensagem em SOAP tem um header e um body.
    - No header estão as definições genéricas, constantes.
    - No body é o que tem o conteúdo da mensagem propriamente dito, difere de mensagem para mensagem.
    - O envelope estabelece o contexto da mensagem, quais são os tipos que podem ser utilizados.

- O soap permite dois estilos de comunicação:
  - ♣ (Remote Procedure Call) RPC-Style
  - ♣ Document-Style

29.

1. **Heterogeneidade** - Capacidade para suportar a variedade e a diferença

2. **Abertura** - Característica de um sistema que determina a sua capacidade de extensão e modificação, a abertura pressupõe a especificação e documentação das interfaces de programação

3. Segurança - A natureza aberta dos Sistemas Distribuídos torna-os mais permeáveis a ataques

4. Escalabilidade - A escalabilidade é a propriedade de um sistema que indica a sua capacidade em responder de forma linear a aumentos significativos da carga, número de utilizadores ou área de abrangência

5. Concorrência- A existência de vários fluxos de execução implica acessos simultâneos a recursos partilhados, torna-se, portanto, necessário garantir a sincronização de certas operações para assegurar a coerência de dados

6. Transparência-A transparência é definida como a capacidade de esconder do utilizador e das aplicações a natureza distribuída do sistema

7. Gestão de falhas –

►Detecção de falhas

♣ Algumas falhas podem ser detectadas

• Ex: mecanismos de checksum detectam erros de transmissão

♣ Outras não são detectáveis

• Ex: crash de um servidor ou congestão na rede de dados?

► Mascarar falhas

♣ Depois de detectadas, algumas falhas podem ser mascaradas

• Ex: retransmissão de pacotes, utilizar cópia local dos dados

► Tolerância a falhas

♣ Capacidade de “tratar” um erro previamente identificado, continuando a oferecer parte do serviço

♣ Especificação do comportamento na ocorrência de falhas • Ex: continuar com funcionalidade reduzida, utilizar redundância

► Recuperação de falhas

♣ Envolve a capacidade de recuperar o estado do sistema antes da falha e reconstituí-lo (roll-back)

♣ Mecanismos complexos que envolvem a noção de transacção

MODELO

1- **Compare as vantagens e desvantagens de um sistema com arquitetura cliente-servidor com uma arquitetura peer-to-peer**

R= No P2P todos os nós (laços) tem os mesmos benefícios, então se um nó falha os outros continuam ativos continuando o trabalho. Já o cliente-servidor se o servidor cai, as solicitações do cliente podem não ser satisfeitas, no entanto um ponto positivo é que como a informação esta centralizada em um só lugar acaba por estar mais segura.

2- **Descreve resumidamente a função do seguinte componente do WSDL: Binding**
R= A função Binding defini o formato da mensagem (associa qual o protocolo de representação)

3- **Quais são os principais desafios dos sistemas distribuídos no que respeita a segurança?**

R= Ao que diz respeito a segurança são mais permeáveis (permite outro passar por ele) devido a como ele é. Sendo assim a segurança dos sistemas distribuídos devem passar autenticação, controle de acesso, entre outros, de forma que possa manter integridade e proteção.

4- **Descreve resumidamente a função do seguinte componente do WSDL: PortType**

R= A função PortType defini os métodos a invocar (é o que define qual interface posso chamar remotamente, de certa forma equivale à interface do java).

5- **Descreva para que serve a seguinte componente do sistema RPC: Interface em IDL**

R= Em termo de desenvolvimento o sistema de RPC precisa de uma Interface Description Language (IDL) esta linguagem serve para que o servidor descreva qual a interface que exporta e para que os clientes possam também entender.

6- **Imagine que sua empresa pretende construir um servidor que exponha os seus serviços através de RPC para o exterior, de forma que outras empresas possam chamar esses serviços. Optaria por expor os serviços em JAVA RMI ou web service SOAP? Porquê?**

R= Para a minha empresa eu optaria em utilizar o web service com SOAP, porque ao utilizar o JAVA RMI teria de garantir que toda a gente está a utilizar o java e ainda assim que seja a mesma versão do java para garantir que o formato da mensagem seja exatamente o mesmo. Ou seja, todos os participantes teriam de usar elementos de tecnologias sincronizadas umas com as outras o que cria uma dependência muito grande.

7- **Descreva para que serve a seguinte componente do sistema RPC: stubs cliente.**

R= Os stubs são códigos que convertem os dados do lado do cliente depois chamam a rotina de RPC para enviar a mensagem, recebem a resposta depois os convertem.

8- **Considere que tem um cliente com SUN RPC usando TCP. Indique qual semântica (Talvez(may-be)**, **Pelo-menos-uma-vez(at-least-once), No-máximo-uma-vez (at-most-once), Exactamente-uma-vez(exactly-once)) que se obtem no seguinte caso:**

Sun RPC com TCP, RPC retornou erro por time out.

- **R=** Se o SUN RPC usando um protocolo com ligação do tipo TCP (dependente de confirmação de envio e receção de pacote) retornou erro por time out a semântica a ser usada é No-máximo-uma-vez (at-most-once), não sei se foi o pedido que não chegou, ou se a resposta não chegou ou se o servidor crashou depois de ter executado a função, então se executou foi no máximo uma vez porque ele não duplica pedidos.

9- **Considere que tem um cliente com SUN RPC usando TCP. Indique qual semântica (Talvez(may-be), Pelo-menos-uma-vez(at-least-once), No-máximo-uma-vez (at-most-once), Exactamente-uma-vez(exactly-once)) que se obtem no seguinte caso:**

Sun RPC com UDP, RPC retornou com sucesso.

R= a semântica a ser utilizada seria pelo-menos-uma-vez (at-least-once), porque tenho resposta então quer dizer que foi executada, mas pode ser que deu time out e eu repeti o pedido.

10- **Quais são os principais desafios dos sistemas distribuídos no que respeita a abertura?**

R= A abertura dos sistemas distribuídos é a propriedade que estabelece a capacidade de ser estendido e implementado novamente. Essa abertura permite, por exemplo, que novos serviços sejam adicionados e disponibilizados para o uso de diferentes tipos de usuários. Presume – se o desenvolvimento de uma interface bem documentada e especificada. Por ser aberto há falhas desse sistema podem ser alvo de más intenções de outras pessoas ou organizações.

TESTE 2

1- Porque precisamos de sistemas de nomeação?

R= Porque os objetos nos sistemas são referidos por identificadores, mas nós humanos temos uma fraca capacidade de memorizar esse identificador e, portanto, nós tendemos a memorizar melhor nomes, então tem de ter um sistema que faça ligação entre os nomes e os identificadores.

De uma forma simplista e abstrata um serviço de nomes não é mais que uma tabela que tem numa coluna os nomes dos objetos e na outra os identificadores dos objetos.

E basicamente tem duas operações:

- **Name binding:** Corresponde em inserir uma linha nessa tabela, e essa linha tem a correspondência entre nome e identificados do objeto.
- **Name resolution:** Dado um nome devolve qual o identificador do objeto. Pode ser feito ao contrário também.

2- **Porque pode ser complicado implementar um serviço para fazer a tradução de nomes para endereços IP?**
R= todos os utilizadores da internet acedendo ao serviço

3- **O que é o espaço de nomeação? Qual é o domínio em que um determinado nome é valido?**
R=

4- **Com endereço IP não consigo aceder a uma máquina, qual endereço preciso?**
R= Dado um endereço IP, no nível abaixo tenho que traduzi-lo para um endereço físico, mark address e é esse endereço que tenho que usar para comunicar com a maquina.

5- **Esta tradução de nomes é normalmente recursiva, hierárquica?**
R= sim, eu vou traduzindo um nome no identificador, no nível abaixo este identificador passa a ser um nome que vou traduzir no identificado de nível mais abaixo e assim sucessivamente até ter um endereço físico que me permita aceder diretamente ao objeto.

6- Os nomes são válidos num determinado espaço de nomeação e, portanto, na maioria dos casos esse espaço de nomeação é um espaço local, como por exemplo os números de telemóvel que é valido apenas em determinado país. Se eu quiser ter um espaço de nomes global eu tenho duas alternativas:
   - **A mais utilizada**: composição hierárquica de nomes.
   - **Outra alternativa**: ter identificadores únicos que sejam logo definidos de uma forma global para o mundo todo.

7- **URL (Uniform Resource Locator):** permite identificar um recurso.
8- **URI (Uniform Resource Identifiers):** começam por dizer qual é o esquema em que este nome se refere, e depois tem o nome de forma hierárquica.

9- **Aliasing**: ter sinónimos (nomes equivales para o mesmo nome).

10- **Concatenar NFS:** posso concatenar nomes, posso pegar em determinada árvore e colocá-la por baixo de outra.

11- Então como é que o serviço de nomeação funciona?

R= A sua implementação esta fortemente baseada na replicação e no cache. Isso significa que na base tenho um serviço de nomes que oferece essas duas funções (bind, loockup) como RPC, e a sua estrutura de dados é aquela tabela de uma coluna com nome e outra com o identificador. E eu posso replicar essa tabela para uma serie de servidores, e é essa forma que isso fica implementado.

1- Porque que o caching e a replicação dos nomes funciona bem no caso dos nomes?

R= replicar significa que aquela tabela que traduz nomes em endereços IP está replicada em vários servidores, e portanto posso fazer essa tradução local. Como o endereço muda muito pouco posso ter a tabela replicada num monte de sítios.

Se eu tiver uma cópia da entrada no servidor local cada vez que acedem a uma endereço vão ao servidor de nomes local e o traduzem imediatamente no endereço ip. Além disso a minha própria maquina pode manter ela própria uma cache local para essas traduções, e portanto traduz da primeira vez e depois nem precisa contactar o servidor de nomes, porque ela própria mantem essa cache local.

Se por acaso o nome não for válido e eu contato uma maquina que não existe, basta eu voltar a contactar o servidor.

2- DNS:

- ✓ Estrutura com domínio de base que permite atribuir o primeiro nível é geriado por um domino global.
- ✓ Eles tem servidores primários (autoridade para fazer a tradução do nome) e secundários (recebem replicas da base de dados do primário)

3- O PROCESSO: O resolvedor de nomes faz um pedido ao servidor de nomes que vai primeiro ao nível da raiz, depois ao primeiro nível e depois ao segundo nível e assim sucessivamente. Isso da primeira vez, da segunda vez o servidor de nomes vai ter a informação já em cache e não vai precisar recorrer aos outros. Os próprios resolvers (maquinas) tem localmente também uma cache se se tiver o nome que quero traduzir já na cache local nem se quer contacta o servidor de nomes. No DNS esses registos se chamam RR (resource register) são pares de nome, valor tipificado, e cada um deles tem um time to live.

4- No caso do DNS para Unix Linux : tem duas componentes

- **Resolver:** vai chamar as rotinas de RPC do servidor, e tem a funcionalidade para manter a cache local e tem ou em memoria ou em disco esta cache local para se já la tiver a tradução, não precisar contactar o servidor.
- **Named:** servidor de nomes. (Oferece sua interface através de RPC).

1- E se a cache estiver cheia?
   Imaginando que a cache tem 100 posições, quando chegar na 101 já não cabe na cache, então vou retirar da cache o nome que estava sem usar a mais tempo e substituir por esse novo.

Se as caches locais do Resolver estão mantidas em memória, quando eu ligo minha maquina tenho a cache vazia, e portanto tenho que contactar o servidor, e se o servidor estiver em baixa não conseguimos aceder a internet.

Protocolo X500= tinha o objetivo de ser realmente universal, e que ultrapassasse algumas deficiências praticas do DNS, uma das coisas meio distorcidas do DNS é que todos os países têm uma extensão menos os Estados Unidos. Mas não teve sucesso a escala global, mas foi feita uma implementação LDAP que vingou a escala local.

Segurança

1- **Quando precisamos de segurança?**
   R= quando vamos partilhar com os outros algo de valor.

2-**Porque o modelo habitual de segurança (DAC) é chamado discrecionario (Discretionary)?**

Porque fica a descrição da aplicação efetuar ou não esse controlo.

2- MAC (Mandatory) cada vez que aceder, o sistema vai verificar qual o nível de classificação do objeto, qual o nível de classificação do cliente, e o agente só vai ter acesso se o agente tiver um nível suficientemente alto.

3- Ataque **Man-in-the-middle** = quando estou tentar comunicar com um servidor, o servidor pode tentar se passar pelo servidor verdadeiro, receber minhas mensagens e depois passar ao servidor verdadeiro, recebe as respostas do servidor verdadeiro e me enviar. O que ele esta fazendo é recolher as informações que estou a passar, e pode chegar a modificar as mensagens que passo ao servidor ou as respostas que me devolve.

4- Ataque **Denial service** = Faço muitos pedidos a um determinado servidor e ele vai "abaixo", como esta sempre a responder meus pedidos, não tem tempo, capacidade, recursos de responder os pedidos dos clientes legítimos.

5- **O que significa cifrar uma mensagem?**
R= Dada mensagem eu cifro com uma chave e obtenho a mensagem cifrada, dada a mensagem cifrada eu decifro com outra chave e obtenho a mensagem original.

6- Os algoritmos de cifra e decifra se dividem em dois grandes grupos

- **Simétricos**: chave secreta, a chave para cifrar é a mesma chave de decifrar, por isso tem de ser secreta, só conhecida pelo emissor e recetor.
- **Assimétricos**: chave publica, uma chave é publica e a outra privada.

7- O algoritmo TEA usa cifra simétrica.

8- Se o ataque acontece como a **repetição da mensagem completa**, resolvemos isso colocando um time stamp para ver se a mensagem é recente ou não.

9- Já se o ataque for por repetição de **blocos de mensagem** (primeiro bloco mostra o time stamp recente os outros dois blocos mudaram a conta bancaria e tem o resto da informação da mensagem) o receptor não pode distinguir que não é uma mensagem legitima. Para resolver isso devemos recorrer a cifra continua ao invés da cifra por blocos, pois a continua depende do bloco original e dos blocos anteriores.

10- O problema do Cipher Block Chaining – Cifra continua é que se eu tiver uma mensagem muito grande ou uma sucessão de mensagens e perder uma pelo meio, não consigo decifrar as mensagens seguintes, então tem que ser reinicializado.

11- A crifra das mensagens garante duas coisas, a sua privacidade e sua autenticidade.

12- Posso garantir a integridade da mensagem calculando um resumo da mensagem e cifro esse resumo com chave secreta e o receptor faz o mesmo processo, calcula o resumo da mensagem e decifra com a chave secreta se obtiverem a mesma coisa quer dizer que a mensagem esta correta. Assinatura depende da mensagem.

13- Na chave secreta: Autenticação serve para ficar uma chave entre o browser e onde esta tentando aceder, por ex: moodle, e essa chave é diferente para cada uma das sessões.

14- Na chave publica: A (agente) e B(servidor). A pode autenticar as mensagens se as cifrar com sua chave privada e quando B receber a mensagem vai decifrar usando a chave publica de A. Sabe se é mesmo de A se a mensagem fizer sentido, se estou a esperar de XML e chega mesmo XML. Se decifro com a chave publica de A e faz sentido quer fizer que a mensagem esta correta porque foi produzida com a chave privada de A, a única entidade que conhece a chave A é A. Por a chave de A ser publica qualquer um pode ler essa mensagem, ou seja é uma mensagem publica.

15- Se eu quiser mandar uma mensagem que só B vai perceber, eu cifro com a chave publica do recetor ou seja B. A envia uma mensagem cifrada com a chave publica de B (tem que ser publica porque A só conhece a sua própria chave privada). E o B vai decifrar a mensagem com sua chave privada. Mas assim não consigo garantir autenticidade (não sei quem enviou) porque a mensagem é publica então qualquer um poderia ter a chave.

16- **Para combinar autenticidade e privacidade:** Primeiro autentico a mensagem assinando com a chave privada de A e depois se eu quiser que ela seja privada eu cifro com a chave privada de B e é essa mensagem que envio pela rede. E do outro lado vou receber essa mensagem e decifrar com ordem inversa primeiro decifro com a chave privada de B e depois decifro com chave publica de A.

17- **No mecanismo de chave publica - chave privada garante flexibilidade, já no mecanismo de chave secreta sempre garante a autenticidade e privacidade.**

18- **Como o mecanismo de chave publica - chave privada é muito mais lento:** Posso usar mecanismo de chave publica - chave privada para estabelecer a ligação com o servidor, enviando mensagens de autenticidade e privacidade e um conteúdos de umas das mensagens iniciais é qual a chave secreta que será usada a seguir, e como só eles sabem, a partir dai começam a comunicar por chave secreta e posso ir alterando essa chave.

19- **Como uma chave pode ser comprometida?**

Vendo a mensagem cifrada e tentando as combinações possíveis de chave, para dada a mensagem cifrada eu chega uma palavra original que faça sentido.

O tempo do ataque por brute force é proporcional a dimensão da chave, e para resolver isso basta colocar chaves grandes.