1.

O clássico problema dos dois generais especifica que dois exércitos, cada um liderado por um general, se preparam para atacar uma cidade fortificada. Um vale separa os dois exércitos e a única forma dos dois generais se comunicarem é por meio do envio de mensageiros através do vale. Infelizmente, o vale é ocupado pelos defensores da cidade e é possível que um mensageiro seja capturado. Os dois generais concordaram que irão atacar, mas não concordaram sobre a hora do ataque. É necessário que os dois generais ataquem a cidade ao mesmo tempo para terem sucesso, caso contrário um único exército será completamente derrotado. Eles devem, portanto, comunicar para chegarem a um consenso relativamente à hora do ataque, e cada general deve saber que ambos chegaram a um acordo. Mostre que num sistema assíncrono tal consenso é impossível de assegurar com qualquer protocolo determinístico com um número fixo de mensagens trocadas entre os generais. Será possível garantir-se o consenso pretendido num sistema distribuído síncrono?

Resposta:

Sistema assíncrono:

- Não é possível:
  1. Não há constrições sobre a rapidez na execução dos processos ou transmissão de mensagens.
  2. Os canais de comunicação, como os que utilizam UDP, não oferecem garantia de entrega, ordenamento ou filtragem de duplicados.
  3. Se uma máquina A envia mensagem para B, se A não receber resposta é impossível saber se a máquina A avariou e não conseguiu processar a resposta ou se a resposta foi mesmo perdida no caminho.
  4. Impossível ter a semântica "exactly-once". Se A retransmite a mensagem e B já tinha recebido e processado, B pode receber a mensagem mais do que uma vez. Apesar dos protocolos conseguirem filtrar duplicados e retransmitir respostas em vez de re-executar operações, a incerteza permanece sobre a última mensagem.

Sistema síncrono:

- O problema continua impossível, mas é possível garantir que os generais tomem decisões diferentes caso a falha de comunicação se deva apenas a atrasos nas comunicações:
  1. Conhecem-se os lower e upper bounds dos tempos para executar as etapas, e cada mensagem transmitida é recebida dentro de um período de tempo bem definido. Além

disso, cada processo tem um relógio local cujo desvio em relação ao tempo real também é bem conhecido

2. A garantia temporal leva a que o problema de distinguir entre falhas do processo ou perda de mensagem seja eliminada. Se A envia mensagem a B e A não recebe resposta dentro de um tempo máximo garantido, A pode assumir definitivamente que B falhou (ou a mensagem foi perdida) e não apenas que está atrasada.
3. Com esse conhecimento determinístico sobre o tempo e a entrega, os generais podem fazer o seguinte protocolo que garante que ambos vão chegar a uma decisão:
    1. A envia uma mensagem de ataque e o tempo ao general B
    2. A inicia um temporizador com base no tempo máximo garantido de ida e volta da mensagem
    3. B recebe mensagem dentro do seu tempo esperado, B envia confirmação a A
    4. Se A recebe confirmação de B antes do timeout acabar, ambos sabem que a comunicação foi bem-sucedida e que o outro está ciente do plano. Então atacam
    5. Se o temporizador de A expirar sem receber a confirmação de B, A pode saber que a comunicação falhou ou que B falhou, então A aborta o ataque
    6. Como ambos operam sob as mesmas garantias de tempo, se a confirmação de B se perdesse, B também perceberia a falta de resposta de A (ou subsequente mensagem de aborto de A) dentro de um tempo limitado, levando a uma decisão consistente de não atacar.

2.

Um cliente faz uma chamada remota a um método transfere(A, B, €1000) num servidor de transações financeiras, usando semântica at-most-once. No entanto, após enviar o pedido, a ligação falha e não recebe resposta, resultando numa exceção. O cliente pode estar certo de que o método não executou? Pode voltar a tentar (retry) de forma segura?

Resposta:

- O cliente não pode estar certo de que o método não executou:
    - O server pode ter executado o método e a mensagem de resposta pode ter sido perdida no canal de comunicação.
- O tanas, esta é uma operação não-idempotente, quando o cliente der por ela, acabou de transferir 2000€ para B, feito parvo. Isto dá-se porque o retry por parte do cliente vai gerar um novo Request ID, mesmo com a semântica "at-most-once" para cada Request ID individual, ele vai considerar o retry do cliente como um pedido distinto.

3.

Como podemos solucionar o problema anterior?

Resposta:

- Anexar chave de idempotência explicita a cada pedido enviado ao servidor, mesmo na camada da aplicação.
- O servidor, antes de executar a operação, deve registar essa chave e o estado do pedido numa base de dados.
- Se o servidor receber um pedido com uma chave já processada, ele só retransmite a resposta original em vez de re-executar a operação.
- Se o cliente quiser dar retry, fá-lo com a mesma chave de idempotência

4.

Por que razão é que todas as operações remotas se traduzem em trocas de byte[]? Como é isto conseguido?

Resposta:

- Para transmitir informação nas redes, é preciso representar dados completos de forma uniforme, para isso é usado o mecanismo de marshalling, que serializa um conjunto de dados, transformando-o num array de bytes,
- No fundo, é uma necessidade técnica para transmissão de dados através de redes.

5.

No contexto das interações cliente-servidor são muitas vezes usados protocolos do tipo request-reply baseados em operações de envio e receção de mensagens. Estas operações de envio e receção de mensagens podem ser descritas em termos semelhantes aos da API de datagramas UDP, apesar de muitas implementações práticas usarem TCP. Descreva três vantagens da utilização de UDP para a invocação remota de métodos usando protocolos request-reply.

Resposta:

- Elimina o overhead dos ACKs, uma vez que no protocolo request-reply, podemos assumir que a resposta do server já funciona como um ACK. Isso torna o mecanismo de ACK tão bom quanto inútil.
- Elimina o overhead do TCP handshake, enquanto o TCP estabelece uma ligação através do processo de handshaking antes de começar a transmitir dados, o UDP dá rawdog.

- Também elimina o overhead do controlo de fluxo do TCP, não é preciso window size e afins. Portanto, em dados de pequena dimensão, o UDP é mais rápido ao não usar essas táticas.

6.

E uma razão forte para se usar TCP?

Resposta:

- Mais confiável, há entrega fiável e ordenada dos dados, bem como filtragem de duplicados:
  - Sequenciamento
  - Controlo de Fluxo
  - Retransmissão
  - Buffering
  - Checksums
- TCP é essencial para apps que não toleram perdas de dados (e.g. FTP, HTTP, SMTP, ...)

7.

Hoje em dia, os stubs e os skeletons existem conceptualmente, embora sejam gerados dinamicamente durante a execução usando reflection em vez de serem pré-compilados. Descreva as funcionalidades que devem suportar, atendendo às questões de comunicação, marshalling, invocações e semânticas.

Resposta:

- Comunicação (Encapsulamento da Rede):
  - Mascaram a complexidade da rede.
  - Gerem abertura e fecho de ligações bem como o envio e receção de mensagens via sockets.
  - São intermediários:
    - Stub (cliente) envia o pedido.
    - Skeleton (server) recebe e encaminha para o objeto remoto (servant).
    - Assim, o cliente/server não lidam com streams ou sockets, isso seria ganda seca.
- Marshalling/Unmarshalling:
  - O stub deve fazer marshalling de pedidos e unmarshalling das respostas.
  - O skeleton deve fazer unmarshalling dos argumentos do pedido e marshalling do resultado ou exceções da resposta.

- Invocação de Métodos Remotos:
  - Fazem com que chamadas a métodos remotos sejam sintaticamente idênticas a chamadas locais!
  - O stub simula a presença do objeto remoto no cliente
  - o skeleton orquestra a execução do método correspondente no servant
- Implementação de Semânticas:
  - Contribuem para implementar semânticas específicas tais como "at-most-once" e afins.

8.

O comportamento típico de clientes RPC (ou RMI) consiste em estabelecerem uma ligação ao servidor e executarem longas sequências de chamadas remotas. Partindo desta observação, que melhoramento podemos fazer ao protocolo request-reply-acknowledge (RRA) para aumentar o desempenho?

Resposta:

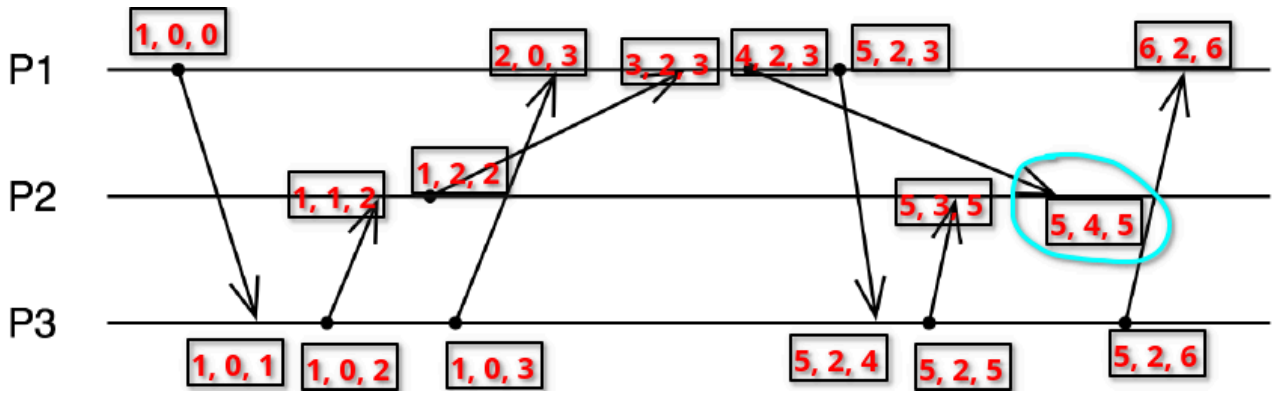
- Podemos agrupar vários ACKs num só packet.
- Também é útil dar piggyback dos ACKs nas mensagens de request subsequentes.

9.

Considere a troca de mensagens entre os processos P1, P2 e P3 representada na figura que se segue. Trata-se de eventos de envio e receção de mensagens ponto-a-ponto.

Usando o algoritmo de vector clocks, indique na figura os vector timestamps em cada ponto de envio e de receção, bem como nas próprias mensagens. Contabilize tanto os eventos de envio como os de receção de mensagens. É possível detetar alguma quebra de causalidade nesta troca de mensagens?

Resposta:

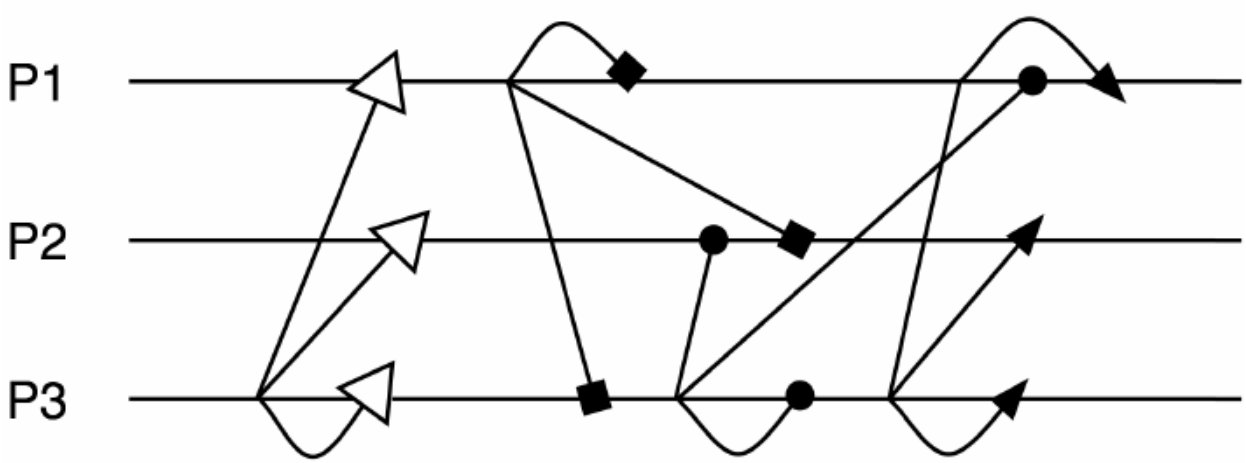


Quebras de causalidade:
Acontecem caso $e \rightarrow f$, porém $VT(e) < VT(f)$ não se verifica.
A quebra de causalidade está assinalada na figura:

- O vetor $\langle 4, 2, 3 \rangle$ chega ao P_2, que tem o vetor $\langle 5, 3, 5 \rangle$

10.

O diagrama que se segue ilustra uma troca de mensagens, através de multicast, entre os processos P1, P2 e P3, sendo que as diferentes formas geométricas representam as quatro mensagens distintas e o momento no qual são entregues à aplicação. Que ordenamento de entrega de mensagens é respeitado?



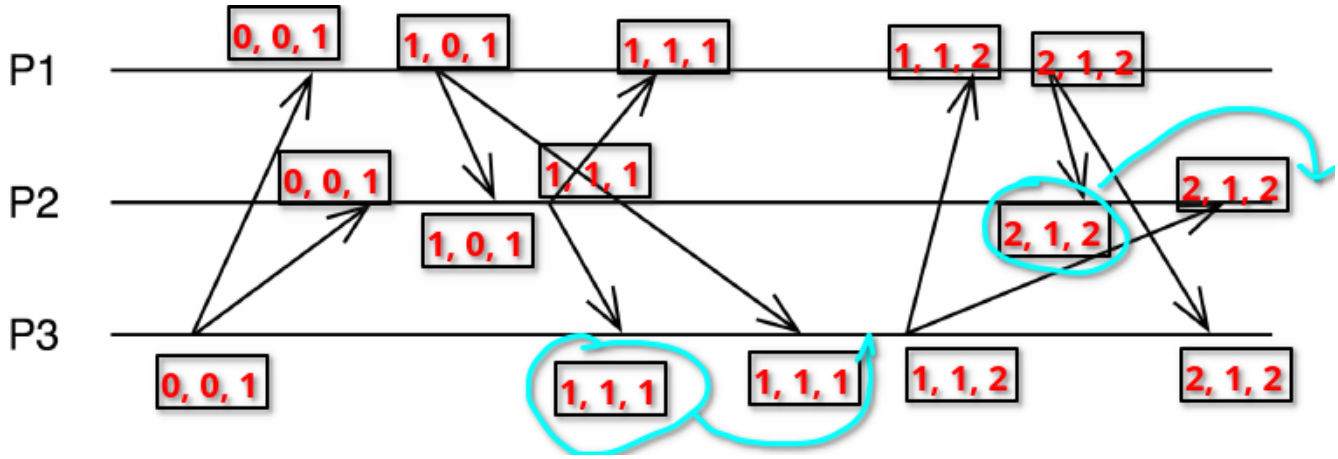
Resposta:

- FIFO, verifica-se pois quando um processo envia uma mensagem $\text{multicast}(g, m)$ e depois $\text{multicast}(g, m')$, os outros processos fazem sempre a entrega de m antes de m'. Mas não há garantia da mesma ordem entre a receção dos diferentes processos.

11.

Considere a troca de mensagens entre os processos P1, P2 e P3 representada na figura que se segue. Trata-se de eventos de envio e receção de mensagens multicast. Usando o algoritmo de vector clocks, indique na figura os vector timestamps em cada ponto de envio e de receção, bem como nas próprias mensagens. Contabilize apenas os eventos de envio. Indique igualmente receções de mensagens multicast que sejam colocadas na hold-back queue e o ponto em que são entregues à aplicação.

Resposta:



12.

Recorde os algoritmos de Cristian e de Berkeley para sincronização de relógios em sistemas distribuídos. Descreva sucintamente as diferenças entre estes dois algoritmos. Explique, em particular, por que razão é enviado o valor absoluto do relógio num caso e o valor do ajuste a efetuar ao relógio no outro caso.

Resposta:

Tipo de sincronização

- Cristian: Externa
- Berkeley: Interna
  Quem é responsável pelo sync?
- Cristian: Server de tempo que responde aos requests
- Berkeley: Master process que faz polling aos slaves
  Quem calcula o offset?
- Cristian: O cliente, com base na hora do clock do servidor e o round-trip-time (RTT)
- Berkeley: O master process, com base na média dos slaves a quem fez polling
  Sensibilidade a outliers e falhas de relógios
- Cristian: Não aguenta falhas do server a que faz a query
- Berkeley: Ignora outliers se tiverem valores absurdos em relação à média. Caso um master falhe, podemos só eleger outro.

No Cristian é enviado o valor absoluto pois o time server não tem conhecimento sobre a latência da comunicação com os clients, nem sobre os clocks dos mesmos, a sua função é apenas fornecer o T_{server} no momento em que a resposta é enviada. É responsabilidade do cliente, que conhece os seus próprios T_0 e T_1 calcular a compensação para a latência da rede (podemos só aproximar a latência com $\frac{\text{RTT}}{2}$).
No Berkeley, o master envia apenas o valor de offset para cada slave para evitar que os

relógios voltem para trás no tempo. Se o valor do offset for negativo, o slave simplesmente abranda a sua clock speed até o ajuste necessário ser feito.

13.

Considere que uma máquina A sincroniza o relógio com uma máquina B executando uma iteração do protocolo NTP no modo simétrico. A máquina A enviou o pedido numa mensagem M_a quando o relógio local marcava $14\text{h}30\text{m}40.500\text{s}$ e a mensagem M_b com a resposta foi recebida pela máquina A à hora local $14\text{h}30\text{m}40.700\text{s}$. A mensagem Ma foi recebida pela máquina B quando o respetivo relógio marcava $14\text{h}30\text{m}40.520\text{s}$. A máquina B enviou em M_b o valor de relógio $t = 14\text{h}30\text{m}40.640\text{s}$ (juntamente com o instante de receção $14\text{h}30\text{m}40.520\text{s}$). Qual será o novo valor de relógio da máquina A?

Resposta:

$$
\begin{aligned}
T_1 &= 40.5\text{s} \\
T_2 &= 40.52\text{s} \\
T_3 &= 40.64\text{s} \\
T_4 &= 40.7\text{s}
\end{aligned}
$$

$$
\begin{aligned}
o &= \frac{(T_2 - T_1) - (T_4 - T_3)}{2} \\
&= \frac{(40.52 - 40.5) - (40.7 - 40.46)}{2} \\
&= -0.02\text{s}
\end{aligned}
$$

O novo valor é $T_4 + o = 14\text{h}30\text{m}40.68\text{s}$.

14.

Relativamente à questão anterior, a máquina B pode fazer a leitura local do valor de relógio t em qualquer momento da execução antes de enviar a mensagem M_b à outra máquina? Justifique.

Resposta:

Não, o cálculo do offset no final assume que a timestamp t foi medida precisamente antes do envio de M_b.

15.

Demonstre que a estimativa da diferença entre o valor dos relógios, obtida no modo simétrico do protocolo NTP, se pode obter através da expressão $\frac{(T_2 - T_1) - (T_4 - T_3)}{2}$.

Resposta:

Seja d o delay da rede em ambos os sentidos e o o offset do relógio de B em relação a A.

Então:

$$\begin{cases} T_2 = T_1 + d + o \\ T_4 = T_3 + d - o \end{cases}$$

$$\Leftrightarrow \begin{cases} d = T_2 - T_1 - o \\ d = T_4 - T_3 + o \end{cases}$$

$$\Rightarrow T_2 - T_1 - o = T_4 - T_3 + o$$

$$\Leftrightarrow o = \frac{(T_2 - T_1) - (T_4 - T_3)}{2}$$

16.

Demonstre que um sistema sincronizado externamente com diferença máxima D está também sincronizado internamente com diferença máxima $2D$.

Resposta:

Encontrar máximo valor de $\mid C_i(t) - C_j(t) \mid$ para quaisquer dois processos i e j no sistema onde t é o tempo real.

- Se um sistema está externamente sincronizado, então:
  - $|C_i(t) - t| \leq D$
  - $|C_j(t) - t| \leq D$
- Então:
  - $\mid C_i(t) - C_j(t) \mid \ \leq \ D + D = 2D_1$

17.

Considere as diretivas de caching existentes no protocolo HTTP, que permitem controlar quais os objetos que podem ser armazenados em cache e configurar esse armazenamento. Descreva sucintamente qual a finalidade de cada uma destas diretivas: no-store, no-cache, private/public, max-age, ETag.

Resposta:

- `no-store` : A resposta não deve ser armazenada de todo, ou seja, o proxy deve remover a informação da memória assim que possível. Usado para informações sensíveis.

- `no-cache`: Força as caches (tanto do proxy como do browser) a submeter o pedido ao server de origem para validar antes de ser usada para uma resposta subsequente. (Verifica a "frescura" do recurso de modo a não retornar info desatualizada, no fundo).
- `public`: A resposta pode ser armazenada em qualquer cache, incluindo as caches partilhadas (como as dos proxy servers).
- `private`: A resposta não deve ser armazenada numa cache partilhada
- `max-age`: Especifica a quantidade máxima de tempo que um objeto será considerado "fresco" (válido para ser servido diretamente da cache sem revalidação). Após esse período, o objeto na cache torna-se stale e, se no-cache não estiver presente, a cache pode optar por revalidá-lo com o server de origem ou eliminá-lo
- `ETag`: Número de versão do recurso. `ETag` é um ID único para uma versão específica de um recurso. é usado em conjunto com pedidos condicionais (If-None-Match) para que o server possa determinar se uma cópia em cache do browser ainda é a mais recente sem ter que enviar o recurso completo novamente. Se o `ETag` do cliente corresponder ao do server, o server responde com `304 Not Modified`, então o cliente pode usar a sua cópia em cache.

18.

Construa um fluxograma que sintetize quais as diretivas de caching a usar em função de: se uma resposta é reutilizável, se deve ou não ser sempre revalidada, se é armazenável por caches intermédias, se tem um prazo de validade, existência de versões diferentes de cada objeto.

Resposta:

1. A resposta pode ser armazenada em cache? `Cache-Control: no-store`:
   - Não é armazenado em nenhum cache.
2. A resposta deve ser sempre revalidada? `Cache-Control: no-cache`:
   - Força as caches a submeter o pedido ao server original para validação antes de usar a cópia em cache.
3. A resposta pode ser armazenada publicamente? `Cache-Control: public`:
   - Pode ser armazenada em qualquer cache, incluindo partilhadas
4. A cache deve ser privada? `Cache-Control: private`:
   - Não deve ser armazenado numa cache partilhada
5. Existe prazo de validade para a resposta em cache? `Cache-Control: max-age=[seconds]`
   - Esta diretiva especifica a quantidade máxima de tempo que o objeto será considerado fresco.
6. Existem diferentes versões do objeto que precisam de ser identificadas? `ETag`:

- Avalia se há várias versões e se é preciso re-transferência do conteúdo completo, faz validação mais eficiente.

19.

Suponha que a Alice pretende enviar um documento D ao Bob, de forma segura. Para tal, a Alice envia ao Bob uma mensagem $M = \{D\}^{K_s}$, $\{\{K_s\}^{K_{B_{\text{pub}}}}, H(D)\}^{K_{A_{\text{priv}}}}$, $\{A, K_{A_{\text{pub}}}\}^{K_{B_{\text{pub}}}}$, sendo que a Alice conhece a chave pública do Bob mas o Bob não conhece a chave pública da Alice (e por essa razão a chave pública é enviada em M).

Notas:

- $\{D\}^{K_s}$ é o documento encriptado com a chave simétrica
- $H(D)$ é o documento hashed
- A é a identidade da Alice
- $\{\{K_s\}^{K_{B_{\text{pub}}}}, H(D)\}^{K_{A_{\text{priv}}}}$ é uma assinatura digital de Alice.

A)

Nestas circunstâncias, a Alice pode estar segura de que exclusivamente o Bob poderá ler o documento? Justifique.

Resposta:

- Sim
  1. O Bob desencripta a $\{A, K_{A_{\text{pub}}}\}^{K_{B_{\text{pub}}}}$ usando $K_{B_{\text{priv}}}$.
     - Obtém, então a identidade de Alice A e a chave pública de Alice $K_{A_{\text{pub}}}$.
  2. O Bob usa $K_{A_{\text{pub}}}$ para desencriptar $\{\{K_s\}^{K_{B_{\text{pub}}}}, H(D)\}^{K_{A_{\text{priv}}}}$.
     - Obtém, então $\{K_s\}^{K_{B_{\text{pub}}}}$ e $H(D)$.
  3. Bob usa $K_{B_{\text{priv}}}$ para desencriptar $\{K_s\}^{K_{B_{\text{pub}}}}$.
     - Obtém, então, a chave simétrica K_s
  4. Finalmente, Bob usa K_s para desencriptar $\{D\}^{K_s}$.
     - Obtém, então, o documento D.
- Tudo isto foi possível graças à chave privada de Bob, que, em princípio, só o Bob é que tem (desde que ele não ande por aí a meter os seus dados privados onde não deve!!!!!!!)

B)

Nesta situação, o Bob pode estar seguro de que o documento D é autêntico e proveniente da Alice? Justifique.

Resposta:

- Para saber se o documento é autêntico, o Bob pode tentar dar hash ao documento D, gerando $H(D)'$ e comparar com $H(D)$.
- Ele não pode ter a certeza que o documento é proveniente da Alice, pode dar-se o caso de haver um man-in-the-middle :/

20.

Considere um cenário no qual a Alice (A) envia um documento X ao Bob (B). As chaves públicas de ambos são conhecidas por todos previamente. A Alice pretende assegurar a confidencialidade do documento e pretende receber uma confirmação de que o Bob garantidamente abriu e teve acesso a esse documento. Descreva com rigor uma troca de mensagens que permita dar essas garantias.

Resposta:

- Alice manda $M_1 = \{X\}^{K_s}, \{K_s\}^{K_{B_{\text{pub}}}}$.
- Bob obtém K_s, lê X e envia à Alice $M_2 = \{H(X)\}^{K_{B_{\text{priv}}}}$.
- Alice obtém $H(X)$ e compara ao seu próprio hashing de X, $H(X)$.

21.

Recorde o protocolo de autenticação de Needham-Schroeder, segundo o qual o processo A pede ao servidor S uma chave secreta K_{AB} para comunicar com o processo B.

1. $A \to \text{KDC} : A, B, N_a$
2. $\text{KDC} \to A : \{N_a, K_s, \{K_s, A\}^{K_b}\}^{K_a}$
3. $A \to B : \{K_s, A\}^{K_b}, \{N'_a\}^{K_s}$
4. $B \to A : \{N_a'^{-1}\}^{K_s}$

A)

Qual é a finalidade do nonce da primeira mensagem?

Resposta:

- Garante unicidade e frescura da transação para Alice.
- N_a é um número aleatório gerado por Alice que assegura que a resposta do KDC não é retransmissão de uma mensagem antiga.

- Ao receber a mensagem 2 do KDC, a Alice vai verificar se o N_a que ela enviou está presente e foi corretamente encriptado pelo KDC, se sim, a Alice sabe que está a responder ao seu pedido atual e não a um replay attack.

B)

Qual é a finalidade da terceira mensagem? Justifique sucintamente.

Resposta:

- $\{K_s, A\}^{K_b}$ é o ticket gerado pelo KDC e encriptado com a chave privada de Bob K_b.
  - Esse ticket contém a chave de sessão K_s que a Alice e o Bob vão usar durante a comunicação entre eles, bem como a identidade de Alice A.
- $\{N'_a\}^{K_s}$ é um desafio para o Bob.
  - Ao incluir o nonce N'_a encriptado com K_s, Alice exige que o Bob prove que consegue decifrar K_s para poder responder a este desafio, confirmando a sua identidade.

C)

Qual é o objetivo das duas últimas mensagens e como é que esse objetivo é cumprido?

Resposta:

- Previne ataques de masquerading e replaying ao autenticar Bob e Alice mutuamente, e ao confirmar que ambos possuem K_s.
- $\{N'^{-1}_a\}^{K_s}$ é a resposta de Bob ao desafio da Alice.
- Alice verifica se o nonce retornado pelo Bob é igual ao que ela lhe mandou. Caso se verifique, ela tem a certeza que a mensagem veio do Bob, pois apenas a chave privada do Bob pode decifrar o ticket e obter K_s, que é usado para obter N'_a.