A-4 What is an anonymity set and why is it important that it is large? Given two anonymity sets AS_i and $AS_j, (i \neq j)$, how would you interpret $AS_i \cap AS_j$?

The anonymity set is a set of entities(people, ip addresses etc) where one entity is considered anonymous within the set. If the anonymity set is not large then one entity in the set can not be considered anonymous to anyone on the outside. Recall Martin's example on the lecture where if one single person would stand on a empty square somewhere and yell out something offensive. Then by-passers hears this yell and looks to the square as it was in the direction of where the yell was coming from. Since there only is one person at the square(analog to entity), i.e. the person is alone in the anonymity set, they will now assume that that person is guilty. This is just an example where there is only one in the anonymity set but the idea works the same if there are few people in the anonymity set.

The intersection between two anonymity sets is the set of entities that reside in both A_i and A_j. The size of the intersection will always be less than or at most equal to the size of the smallest set used in the intersection. I.e. when you keep taking intersections between different anonymity sets where you know that one certain entity resides in both sets, then the resulting set from the intersection will (most likely) get smaller and smaller for every intersection and you may eventually be able to identify that certain entity.

A-11 When returning a message using an untraceable return address, why does each Mix encrypt the return message with the randomness R_i?

When client x wants the recipient y to be able to respond to him/her with a untraceable return address he creates the random string R_1. This number together with real address of x, A_x is then encrypted with the public key of the last mix in the chain, K_1 which leads to the encrypted message $K_1(R_1, A_x)$. This encrypted message is sent as a part of the real message that client x sends to client y. So client y receives $K_1(R_1, A_x)$ inside of the message M that client y received in a $K_y(R_0, M)$ packet. The real return address of client x is encrypted and is therefor unknown to client y, all is well. So when client y wants to respond with a new message M_y and encrypts it to $K_x(R_0 y, M_y)$ and uses the received encrypted return address as the return address, forming $K_1(R_1, A_x), K_x(R_0 y, M_y)$ and sends this to mix 1. Mix 1 is then able to decrypt the return address using its private key obtaining $R_1, A_x, K_x(R_0 y, M_y)$. Mix 1 will now encrypt the message again using R_1 which yields $A_x, R_1(K_x(R_0 y, M_y))$. Now only client x can decrypt the output because he/she created R_1 and is in possession of K_x. The address part $K_1(R_1, A_x)$ should never be the same so that unlinkability is still true, client x must therefor supply a new return address(with a new random string R_1) for every mail he/she wishes to get a response to. It prevents adveraries from simply guessing the return address and verify the guess using the

public key of the Mix. It also prevents an eaesdropper to compare the messages entering and elaving the mix. Without R_1 the message $K_x(R_0, M)$, would be seen on both sides of the mix.

Short answer: To ensure that only the sender of the original message(client x) is able to decrypt and read the response from client y. The sender will add random values $R_1 \dots R_{n-1}$ in the original message so that if Mixes are cascaded they will append a layer of encryption with one of these R values each.

A-12 Regarding replay attacks on Mixes, two protections are suggested in the lecture notes. Which? Would you say that any of them is the better choice? Show how the two strategies can be combined and how this can make the protection more efficient

The first mentioned protection is to save a hash of each received message and for the mix to not allow duplicates by checking if the hash has already been received, and then discarding the message if that is the case. This would require the mix to have a large memory drive to save all this information and the space requirement would just get higher as a function over time. Another option is for the history of all messages to be kept in every message resulting in large overhead for every single message but saves the memory requirement on the actual mixes.

The second protection is to include a timestamp in the message in order to verify that the message is fresh and control the validity of the message. The timestamp needs to contain either implicitly or explicitly the start time of the interval and the duration, or the equivalent start and end time in order to work.

A way to combine these strategies would be to send the actual timestamp, the message, and a hash of the message concatenated with the time stamp, $h(M||t)$.
When the Mix receives the message it can just check that the message is fresh by comparing the received time stamp to the current time, and then compare that the hash of the message and given timestamp matches the hash received. This hash is then saved for a timestamp period and messages with equal hashes can be disregarded.

A-17 When Alice creates a Tor circuit, who selects the relays that are used?

The Tor network chooses the relay nodes for Alice(via the Tor client she has on her computer). Tor network have specific selecting rules (seen in section 2.2 Path selection and constraints) of the Tor spec. The path for each circuit is chosen before the circuit is build. The exit node is chosen first followed by the other nodes in the circuit. All paths obey constraints that again can be seen in section 2.2 of the Tor spec.

A-24 Alice is negotiating keys during a chain construction in Tor. It is reasonable to assume that sending material to and back again from OR_1 takes some time. Can she use this time to prepare for negotiating with OR_2, OR_3, ...? How/why not?

The setup requires public key operations which takes time to compute. Before any data can be relayed to a node a symmetric key with that node must be negotiated. So Alice could negotiate K_1 with OR_1 and while she is waiting for the response from OR_1(to get the symmetric key K_1) she could actually prepare for the key exchange with any other node OR_n down the circuit by computing $K_{OR_n}(g^{x_n})$. But Alice has to wait until she gets the response symmetric key K_{n-1} from OR_{n-1} before she can actually start the key exchange with node OR_n as Alice needs to encrypt the messages with the previous nodes symmetric key(which has been negotiated), $K_{n-1}\{K_{OR_n}(g^{x_n})\}$.

A-25 Explain what the point of the recognized field in a Tor cell is and how it makes communication more efficient.

The recognized field in any unencrypted relay payload is always set to zero. That is when the recognized field is zero and the digest of the packet is correct then the cell(packet) should be interpreted at the current OP, otherwise it should be relayed to next OP in the circuit.

This makes it time efficient to check if you should interpret the cell or relay it by just checking the field and running one single digest.

A-26 A TCP handshake consists of the client and the server exchanging three messages: SYN, SYN-ACK and ACK. Explain why, in Tor, Alice can connect to a webserver and expect the TCP handshake with the server to be performed with low latency.

The handshake is done between the connecting party and the actual server. In the Tor network the actual handshake is done between the exit node of the circuit and the server. There is no hops in between a exit node and a server and is therefor not delayed and works as a normal direct TCP handshake between Alice and a server if she would not use a anonymity network. The delay that you get when you use TOR network comes from that the nodes inside the network might have a big geographical distance(one node might be in Germany, other node in Japan etc) between them and therefor cause a delay in the traffic.

A-28 Show that the SSL/TLS handshake, when RSA is used, does not provide perfect forward secrecy

A RSA key has the public modulus n and exponent e and a private exponent d. This key is normally a long-term key where the parameters don't change in a long time. A SSL/TLS handshake contains the following steps:

1. Client sends a Hello message, containing a random number(or byte string to be exact), a, together with some other information(like prefered CipherSuites etc).

2. Server responds with a Hello message, containing a server chosen random number, b, together with the chosen CipherSuite, session ID and the servers digital certificate. Its optional but the server may also add in a client certificate request in this message.

3. The client now verifies the servers digital certificate by checking the digital signature, checking the certificate chain, the expiry and activation dates and the revocation status of the certificate

4. The client now sends a new random byte string, the session key, which is encrypted with the serves public key

5. If the server sent a client certificate request the client sends a random byte string encrypted with the clients private key, together with the clients digital certificate or a no digital certificate alert.

6. The server verifies the certificate in the same way mentioned above.

7. The client sends a finished message, encrypted with the secret key(which was sent encrypted in step 5)

8. The server sends back a finished message, encrypted with the same secret key

9. Now the handshake is complete and "secure" communication can begin by encrypting messages symmetrically with the shared key

So lets say that a adversary records all of your SSL/TLS sessions for a long time and then gets the hold of the server private key. The adversary is then able to decrypt the secret that was sent in step 4 and is now in possession of the session key, which can be used to decrypt all data sent in that session. This can be repeated for all previous sessions that the adversary recorded and therefor gives no perfect forward secrecy.