Crypto Questions and Answers:

Symmetric Crypto:

1. **How do you actually create a new secret key for OTP each time? Is it just a binary string that is random? Does the computer do it, or can people do it manually?**

The important things for the One-Time Pad (OTP) are that the secret key is truly random, as long as the message that you want to encrypt and that it is used only once.

For the Vernam OTP, the characters in the message and in the key are binary (0's and 1's), and the operation to combine plaintext and key and generate ciphertext is just XOR. This means that you need a truly random binary string with the same number of bits as the message you want to encrypt. And you need a new, different key for each message.

For this to be useful, you need to be able to:

a) generate a truly random binary string of the required length, and
b) have it available for both encryption and decryption.

For the generating, you need a source of randomness. Most programming languages have functions to provide *pseudorandom* output (something like rand (for c), Random (for Java), random for python), but these are NOT truly random. They are entirely deterministic, (so not random at all) but they produce sequences that *look like they are random*. But that's not good enough to meet the conditions for the OTP. There's an easy to read discussion of the difference between random and pseudo random numbers in wikipedia: http://en.wikipedia.org/wiki/Random_number_generation#.22True.22_random_numbers_vs._pseudo-random_numbers, and also of some of the methods for obtaining random outputs: flipping coins, rolling dice, measuring radioactive decay, etc. these are things people could do manually, but it may be quite slow.

Making the key available for encryption and decryption requires a secure means of either storing the key (if the data is encrypted in storage and will be retrieved later) or sending the key (if the data is encrypted during transmission over an insecure channel to a receiver, who will then decrypt). But if you have a means for storing the key securely, or transmitting the key securely, why not use this for the message (which is the same length), and forgo the encryption?

2. **What does a time varying function mean in our context?**

This concept relates to stream ciphers, and is about the difference between block ciphers and stream ciphers. *Block ciphers* divide plaintext and ciphertext into blocks, and perform encryption and decryption on blocks of data (typical block sizes are 64 or 128 bits). The standard block cipher mode (ECB) treats each block independently. This means, given a particular key, if a block of plaintext occurs multiple times, then each time it will produce the same ciphertext block. The function doesn't vary over time. For *stream ciphers*, encryption and decryption is performed one character at a time. The character size is usually a bit. If the same character is repeated in the plaintext, we don't want it to produce the same character in the ciphertext. We want the output to vary over time. Time here actually refers to position in the bitstream: the first bit, the second bit, the third bit, etc. For binary additive stream ciphers, the keystream is the critical thing in ensuring this. It should not be repetitive or predictable. A truly random binary string is the perfect keystream to use (aside from the practical issues with generating, storing and distributing it ☺). But if you can't have that, you might use a pseudorandom binary string. Some pseudo random keystream generators are better than others. The pseudorandom functions in most programming language libraries are not good enough for cryptographic purposes.

3. **Once you've encrypted a message, if the ciphertext is modified (say, a bit is 'flipped', or deleted or inserted) when you decrypt the recovered plaintext will be affected, right? How badly could it be affected?**

This depends on both the type of encryption and the type of error.

If it is a binary additive stream cipher; a bit flip will cause the corresponding recovered plaintext bit to be flipped. So the original plaintext and the recovered plaintext will be exactly the same except for in that one bit position. However, if a bit is inserted or deleted, then you lose synchronisation between the keystream and the ciphertext. From that point (where insertion or deletion occurred) onwards the recovered plaintext and the original plaintext will be different (so the original message cannot be recovered), but before this point it will be exactly the same as the original plaintext.

If it is a block cipher in ECB mode then for a given key, a particular plaintext block is encrypted to form a particular ciphertext block (and that same ciphertext block is decrypted to give the original plaintext block). If there is a bitflip error in the ciphertext then you are decrypting a different ciphertext block, so you will recover a different plaintext block. That is, the whole block will decrypt incorrectly.

For other block cipher modes (there are lots, but we only talked about ECB mode and CBC mode in the lecture) modifications in the ciphertext may result in different changes in the recovered plaintext.

Asymmetric Crypto:

1. **With asymmetric ciphers, is there a standard key to disclose? That is, is it the encryption or decryption key that is made public, or does it vary?**

For asymmetric ciphers, participants have a key pair. One of the keys is made public and one is kept private. That concept is standard. Now, which of the keys is public and which is private? That depends on what you are using the asymmetric cipher for. If it is for:

- *Confidentiality* then the encryption key is made public (so anyone can use it to encrypt a message and send to you) and the decryption key is private (so only you can decrypt the message).
- *Signing* then the signing key is private (so only you can sign messages) and the verification key is made public (so anyone can use it to verify your signature on a message they've received).

2. **If someone finds out your private key, is it easy enough to change it?**

If someone finds out your private key, and you have used an asymmetric cipher for

- *Confidentiality* then the decryption key can be used to decrypt messages encrypted using your public key and sent to you (so the attacker can now read these messages)
- *Signing* then the attacker can use your private key to sign messages they claim are sent by you; others can use your public verification key to verify the signature on a message they've received and will be assured it is from you (even though it's actually from the attacker pretending to be you).

So you no longer have the security services you were using the crypto for. Now you need a new key pair. The commands to generate keys depend on the purpose and algorithm. See http://msdn.microsoft.com/en-us/library/windows/desktop/aa388212(v=vs.85).aspx or https://help.ubuntu.com/community/SSH/OpenSSH/Keys or http://docs.oracle.com/cd/E19253-01/816-4557/sshuser-33/index.html for examples.

Also, you need to let people know that the old key pair is not to be used anymore. That is, you need to revoke the public key which corresponds to the compromised private key. If your public key has been certified by a CA then the CA can revoke the certificate. CAs publish Certificate Revocation Lists. If it is on a keyserver (like the MIT PGP keyserver), then you can create a revocation certificate and upload that to the server where the key is stored. See http://pgp.mit.edu/faq.html for example.

3. **If an attacker alters the key to one they know, would they only be able to see the message if their key is actually the same as the original key**?

When an attacker alters a user's public key, they usually replace the existing public key with a different public key value – one for which they know the corresponding private key (ie they've generated a key pair and used the generated public key to replace the existing public key, and kept the generated private key secret). So now the attacker can use the private key to decrypt messages encrypted using the published public key. However, the user whose public key has been replaced will not be able to decrypt any of the messages encrypted using the new public key: their private key only worked when paired with the original public key (which has now been changed). Similarly for signing with the new private signing key.

Key Management:

1. **With keys in general, how are they actually distributed? Would the user physically see the keys contents? Otherwise, how does this happen?**

This is an important part of using cryptography. If it is symmetric crypto, the key needs to be available at encryption and decryption. If you use symmetric crypto for communications then the shared secret key is needed at both sending and receiving ends. So you have to get it there. But you can't send keys as plaintext: anyone who finds the keys can decrypt information encrypted using them. So you have to be able to distribute them without compromising the confidentiality of the key. We had three approaches in the lecture:

i. predistribution,
ii. send them encrypted by another key (either the keys shared with a TTP, Kerberos style; or using asymmetric crypto) or
iii. derive a shared key (Diffie Hellman) by sending components that don't compromise the actual key, and then the participants construct their key from the components.

2. **How do asymmetric ciphers greatly simply key distribution?**

The big simplification is that the public key has to be distributed but it doesn't require confidentiality so you can just send it to people, or post it on a website, etc. The private key does require confidentiality, but it doesn't need to be distributed. It does need to be stored securely, because you don’t want anyone else to find out your private key (see the answers above). And these are large integers (typically a couple of thousand bits) so you are not going to be able to remember this. It will need to be stored. So you might need to store the private key encrypted (symmetric encryption) but you will need to decrypt it in order to use it.

4. **Are there examples of modular exponentiation equations that could work (with actual numbers) with Diffie-Hellman? Also wouldn't this be easy to break from an attackers point of view, as all they need to do is write a script to solve equations?**

There’s a nice explanation of Diffie Hellman on YouTube, that starts with the concept and then includes examples with integers: http://www.youtube.com/watch?v=YEBfamv-_do

Of course, the examples that we follow are for small integers (3, 17, etc) and for these it is possible to try every possibility and figure out what Alice and Bob used as their secrets (the exponents). For secure schemes, the numbers used have to be big enough so that the time to try all possibilities is impractically large. The discrete log problem is solvable in a reasonable time period for primes (used as the modulus) of around 160 decimal digits (more than 500 bits long) so you should use primes of over 1000 bits in length. These are pretty big numbers!

The same mathematical theory is used for asymmetric encryption in the ElGamal scheme.