red - idk at all -- most importante
blue - reasoning's kind of shaky
would appreciate if all answers, especially towards the bottom, were verified
might post on Piazza for everyone
IF SOMETHING DOESN'T MAKE SENSE, CALL ME OUT IN A COMMENT
note: this blasphemy was contributed to by a bunch of nerds. props to all of them

8.

a.

i. **If you don't trust your server, you cannot use OpenSSH to connect to it, as it can easily corrupt your client.**
   - false -- if you don't trust your server, you can't use OpenSSH to connect to it b/c, when client is trying to verify server, it sends a message signed with the server's supposed public key
     server should then decrypt that verification message with its private key and send it back to the client
     so the reason this statement is wrong is b/c of ITS reasoning (that it can corrupt your client)
   - interpretation:
     - "your client" = your computer
     - untrustworthy (malicious) server is one that alleges that it's got another public key

ii. **If you don't trust your client, you can still use OpenSSH to connect to a trusted server.**
   - false -- if you don't trust your computer (someone might be using a keylogger on it, or they could change the info you send over), you can't/shouldn't use open OpenSSH to connect to a trusted server (attacker can get your username and password)

iii. **If you don't know the name or IP address of your server, you can use OpenSSH to discover this info in a secure way.**
   - false -- example: we need the hostname to get onto linux servers and the IP address to get onto our beaglebones

iv. **If you don't know the name or IP address of your client, you can still use OpenSSH to connect to a server.**
   - true -- example: we don't need the name or IP address of our computers to log onto linux servers

v. **If you don't trust your network, you can still use OpenSSH to discover whether your server is running.**
   - false -- defeats the purpose of OpenSSH -- nothing sent over network hidden anymore

b. background about SSH-agent at bottom

i. **ssh-agent improves security by making a copy of private keys.**

- true -- You no longer need to retype your password each time you log onto server for as long as you're logged into your client/computer.
-

ii. **ssh-agent acts on your behalf by running on the server and executing commands there, under your direction.**
- false -- just a fact (full info in background portion at bottom)

iii. **Even if the attacker surreptitiously replaces the ssh-agent program with a modified version, your communications will still be secure.**
- false -- your private keys be caught yo

iv. **ssh-agent eliminates all need for password authentication when communicating to the SEASnet GNU/Linux hosts.**
- false -- still need to type in password once for SSH-agent to load your private keys into memory

v. **If you successfully use a typically-configured ssh-agent and then log out from the client and then log back in again, you can then connect to the same SSH server again without typing any additional passwords/passphrases.**
- true -- just a fact (full info in background portion at bottom)

c.

i. **OpenSSH typically uses public-key encryption for authentication, b/c private-key encryption is less secure.**
- interpretation: private-key encryption is symmetric key encryption (there's no such thing as asymmetric key encryption where you sign with your private key and others decrypt it with your public key -- asymmetric key encryption only works in one day -- lock with public, unlock with private)
  - symmetric = private key encryption b/c keys only known to sender and receiver (compromised otherwise)
- false b/c symmetric key encryption is just as secure as public-key encryption -- problem is key distribution

ii. **OpenSSH typically uses private-key encryption for data communication, because public-key encryption is less efficient.**
- true
  - symmetric key encryption is indeed more efficient than asymmetric key encryption since it works both ways -- easier to decrypt. meanwhile asymmetric key encryption is such that it only works one way -- takes much more complex math to achieve

iii. **When you run 'ssh', it chooses its authentication key randomly from a large key space to make eavesdropping harder.**
- false -- it builds the authentication key from scratch

iv. **The OpenSSH client and server are essentially symmetric, so that it's easy and common to use the same program as either a client or a server.**

- false -- as part of setting up the lab, might've needed to download OpenSSH -- had to run 2 different install commands - implies different programs

v. **Once your private keys are 1024 bits long, there's no point making them any longer, as they're impossible to break.**
   - false -- please, our computers are really good at brute forcing these sorts of problems -- no one is safe

d.

i. **OpenSSH is not limited to just one client-server connection; for example, a team of 5 people can use OpenSSH to communicate information to each other.**
   - true -- ex: can DM people on the linux servers

ii. **For security, OpenSSH refuses to connect to programs written by other people; for example, a client running the OpenSSH code will connect only to a server running the OpenSSH code.**
   - This is false, think of the process of viewing websites on a web-browser. You can successfully view these sites across different browsers like Chrome, Firefox, Safari. The only requirement is that your browser should support certain protocols required for secure transmission, the https that you see in web-page addresses is a protocol used to transfer webpages, in a similar way OpenSSH is one program that implements the SSH protocol. Other programs that implement SSH protocol can also be used to establish a secure session.
   - Some examples and finer details of what specifically needs to be supported here: https://www.openssh.com/legacy.html

iii. **Although port forwarding can be used to display from an OpenSSH server to an OpenSSH client, the reverse is not possible: you cannot use port forwarding to display from an OpenSSH client to an OpenSSH server.**

iv. **For security, port forwarding cannot be chained: that is, you cannot ssh from A to B, and then from B to C, and use port forwarding to let a program run on C and display on A.**
   - false -- ex: we ssh from our computers (A) to our Beaglebone (B), to our partner's Beaglebone (C) then called firefox or xeyes on there, which displayed on our computers (A)

v. **When using port forwarding when connecting to SEASnet, one should take care not to create a forwarding loop, as this can lead to a cycle of packets endlessly circulating on the Internet.**

e. background about making and verifying signatures at bottom

i. **It's not a good idea to connect to a SEASnet GNU/Linux server and use GPG on the server to sign a file, because then an attacker on the network can easily snoop your GPG passphrase.**
   - false -- whole point of OpenSSH is that it encrypts everything going over the network so any snooping is pointless

ii. **A detached signature file must be protected as securely as the private key it's based upon; otherwise an attacker will be able to forge your signature more easily.**
    - false -- if this were true, there'd be no point in sending your detached signature file and supposedly leaving yourself vulnerable to fraud attack

iii. **When generating a key pair, it's important to use a private entropy pool on SEASnet, not the shared pool that everybody can access, because otherwise an attacked on SEASnet might be able to guess your key more easily by inspecting the public entropy pool.**
    - false -- that entropy pool is fueled by a bunch of commands that a significant amount of random people run at random times -- you'd be wasting your time trying to find a pattern in all that

iv. **Exporting a GPG public key to ASCII format neither improves nor reduces its security.**
    - true -- changing it to ASCII doesn't change the actual data...?

v. **Because a detached cleartext signature isn't encrypted, it is easily forged by an attacker with access to the file being signed.**
    - false -- the signature's generated by a combination of your private key and the file being signed, so it's not easy to forge. Doesn't matter what state the signature is saved in.

SSH-agent (courtesy of Rishab Doshi):

- key-based authentication (as opposed to password-based authentication)
    - client's private key used by SSH program to decrypt challenge msg sent by server
    - SSH prog reads private key stored in filesys
        - private key stored on filesys in encrypted format to thwart attackers w/ access to filesys (and therefore access to client's private key)
        - encrypted with passphrase typed in by user while creating public key
        - type in password -> private key decrypted -> ssh uses it to decrypt challenge msg -- get onto server and emacs your heart out
    - gets annoying to be asked password every time client logs onto server so:
        - program called SSH-agent loads in private keys to give to SSH client to answer challenge msgs
        - client just types in password once to load private keys in, then don't need to type in password for rest of login session on your computer/client

Making and verifying signatures summary:

- process different from encrypting w/ receiver's public key and decrypting w/ receiver's private key
- sender can run it through message digest algorithm ("summary" of msg), and sign with own private key (encrypt using private key) to create digital signature
    - if don't want to modify msg, clearsign it (wrap msg in ASCII-armored signature)

        - however, still need to edit signed doc to get actual original
        - to really not modify msg, send separate detached signature
        - gpg --verify just checks if detached signature corresponds to doc
            - problem: it could say "Good signature from Eggert" but while creating my keys and stuff, I could've inputted my name as "Eggert"
            - only way to be sure this is truly from a certain person (can't rely on name alone) is with digital certificate containing sender's public key and variety of other ID info (issued by Certificate Authority)
- receiver gets msg and recovers original msg from digital signature using sender's public key
    - receiver takes received msg (the actual doc rather than the encrypted form, aka the digital signature), runs message digest algorithm on msg to see if it matches decrypted digital signature
        - tells whether anything was tampered along the way