UCLA CS35L

Week 2

Monday

Reminders

- HW1 due tonight!
- We are going over HW2 in class this week, and it is due next Monday (April 13)

- Anonymous feedback for Daniel - https://forms.gle/tZwuMbALe825DBVn8

Focus this week

- Review the inner workings of SSH, and why it is important
  - SSH is how we connect to the school linux servers, so it's worth investigating what is happening in the background
  - It's likely in your career you will use SSH to connect to other servers

- To discuss SSH, we need to talk a little about:
  - Cybersecurity and Encryption
  - Networking

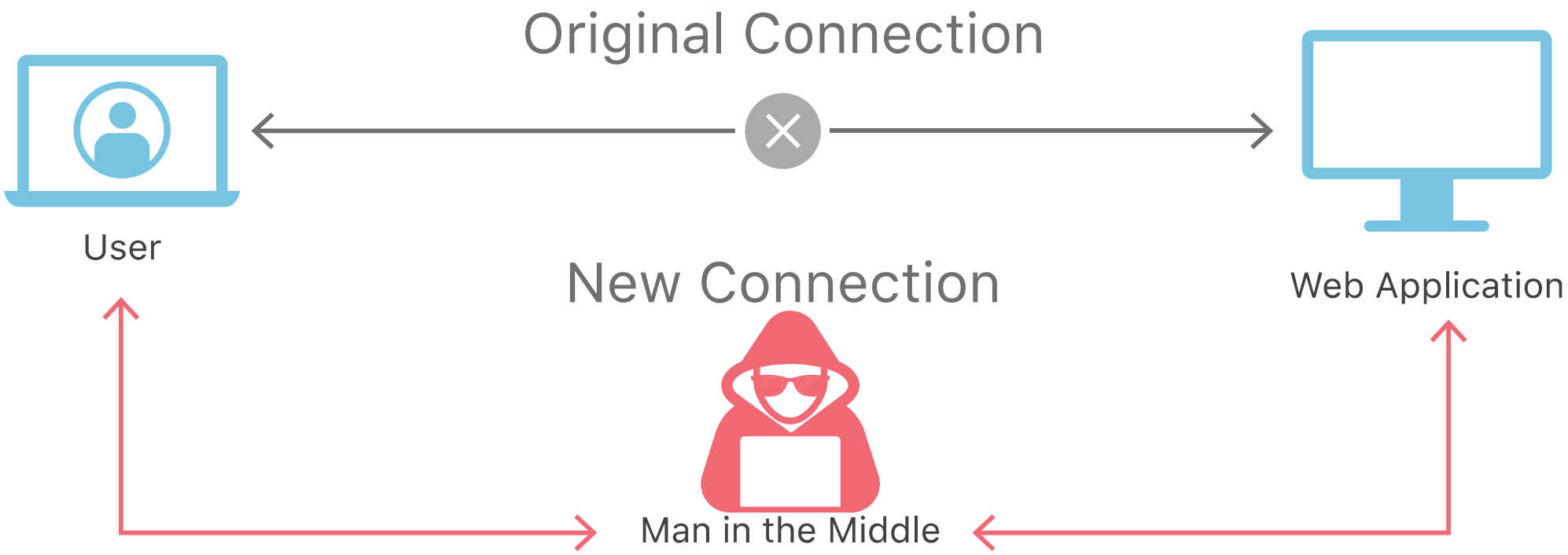
- DISCLAIMER – This is not a cybersecurity or networking class. So we are going to stay at a high-level, just enough to understand how we use these concepts in a tool like SSH

Communication in the Dark Ages of the Internet

Unencrypted communication

- We used to visit websites, download files, log into servers and it was all unencrypted plaintext! Protocols like:
  - TELNET
  - FTP
  - HTTP
- What does plaintext mean?
- Why is this a problem?

Sniffing and Man in the Middle



The beginning of crypto

- What are the goals of crypto?
  - Confidentiality – Message Secrecy
  - Integrity – Message consistency
  - Authentication – Identify confirmation
  - Authorization – User access is specified and controlled

- We are going to focus on SSH and Encryption, and how that facilitates the above.

SSH

- Developed in 1995 by Tatu Ylonen (researcher in Finland) as a defense against password sniffing.
  - Before, everyone logged on to servers with unencrypted protocols like TELNET
- Went through version changes, etc and the SSH-2 protocol became the most popular version after it was made into OpenSSH.

- Designed as a secure way to administer servers
  - Encrypts communication between client and server with public/private keys
  - Enforces authentication with a sign-on process

Encryption

Types

- Symmetric
  - There is a shared secret
  - The same key is used to encrypt and decrypt data
- Asymmetric
  - 2 different (but closely related) keys – Public and Private
    - Only the creator knows the relation. The Private Key cannot be derived from the Public Key
  - The Public Key is shared with everyone
  - The Private Key is only stored locally, and never shared.
  - The Public Key encrypts data that the Private Key can decrypt and…
  - The Private Key encrypts data that the Public Key can decrypt

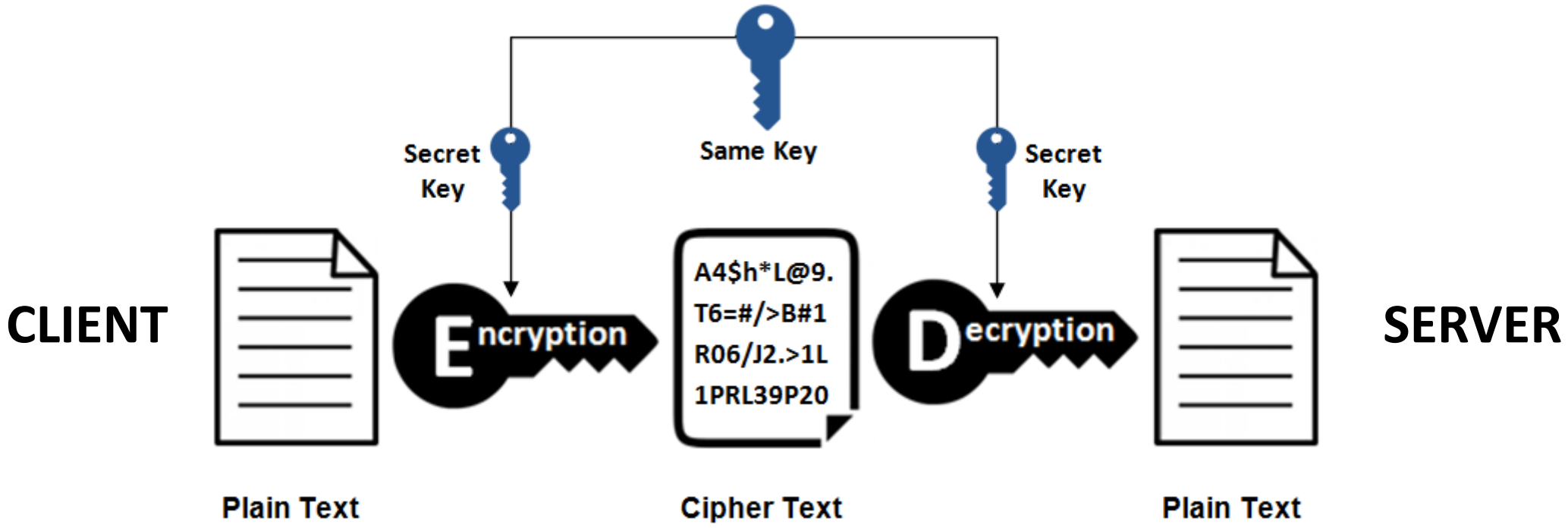
Symmetric Encryption

- Same secret key used for encryption and decryption
- Recommended Industry standard – AES256

- Example – Word substitution

- What is the biggest problem with Symmetric Encryption?
  - Key distribution
  - People who are trying to listen in, will figure out the key when you share it with someone

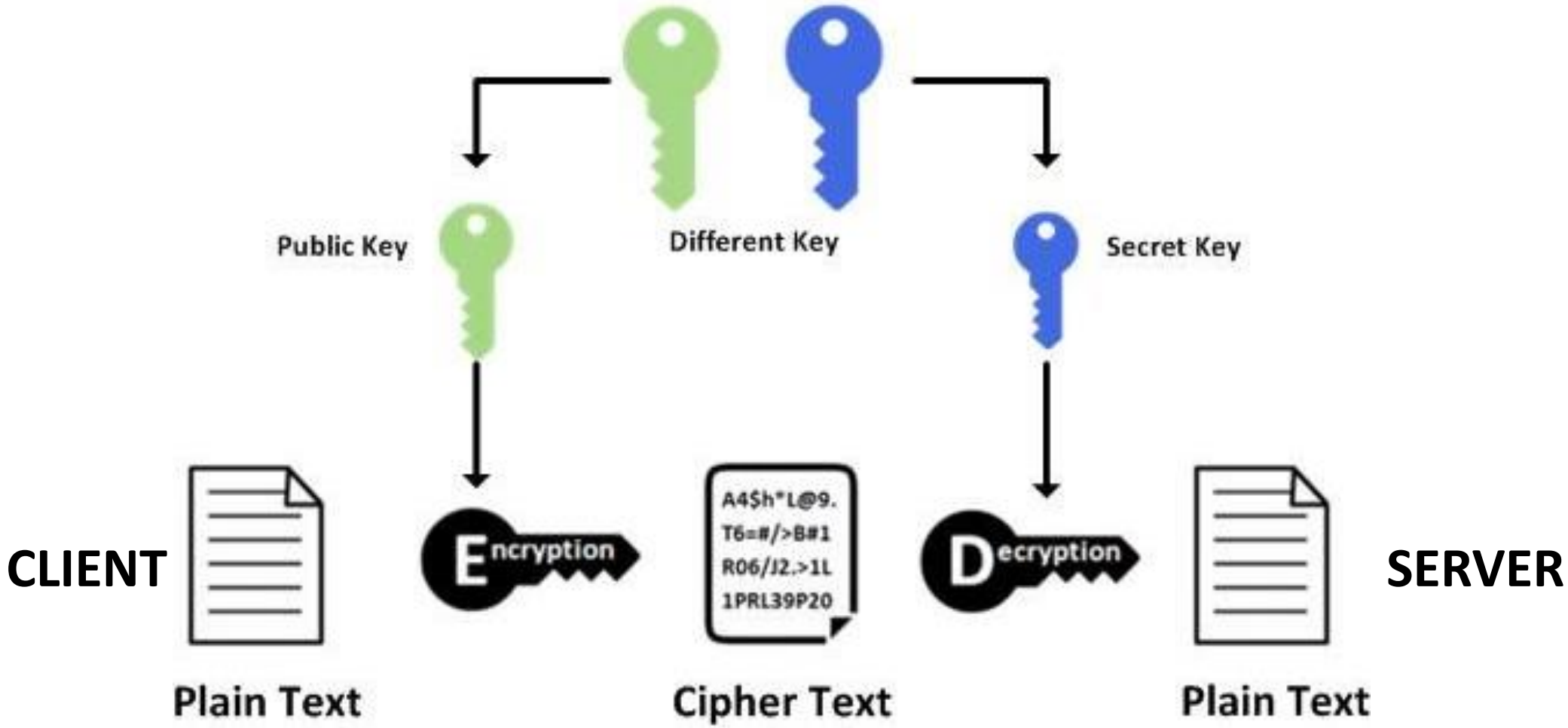
Symmetric Encryption



Public Key Cryptography – Asymmetric Encryption

- Uses a pair of keys to encrypt
  - Public Key – Published and shared with everyone
  - Private Key – Secret Key known only to the owner
- Encryption
  - Clients use the public key to encrypt their message
  - Only the server can decrypt their message with the private key
- Example – Colors
- Industry Used Algorithms
  - RSA - Large Number Prime Factorization
  - DHE – Discrete Logarithm Problem

Asymmetric Encryption



Colors Example

- Watch Youtube Video linked below from 2:20 until 3:50
  - https://youtu.be/wXB-V_Keiu8?t=140

- Ultimately the example above just uses colors. But you can continue watching the video to see an example with actual numbers.
- This video shows a way to exchange keys called RSA
- There are other asymmetric techniques though, DHE is another common one
  - DHE relies on the discrete logarithm problem

General Structure of Secure Connections

- Use Asymmetric Encryption (Public/Private Key) to share the Symmetric Key
- Symmetric Encryption used for rest of the data transfer
- Asymmetric is much slower and performance intensive than Symmetric

- In case you are curious, the internet also uses:
  - Certificates are combined with Public Keys. These certificates announce the identity of the website, and show you have been verified by a central source
  - The Certificate is ‘signed’ by the private key, and we use the public key to verify that signature

Generalized Encryption Functions

- At a high-level then, for secure communication we can imagine the following 4 functions:
  - Encrypt(plainText, publicKey) → cipherText
  - Decrypt(cipherText, privateKey) → plainText

  (Will discuss signatures more on Wednesday)
  - Signature(data, privateKey)
  - Verify(data, signature, publicKey)

How does SSH Work

- SSH encrypts user communication to server, and verifies host identities with signatures.

SSH Server Verification

- First step – server proves it’s identity to the client
  - If its your first time talking to the server, you will get a message like the below.
    - The authenticity of host 'somehost (192.168.1.1)' can't be established.
      RSA key fingerprint is 90:9c:46:ab:03:1d:30:2c:5c:87:c5:c7:d9:13:5d:75.
      Are you sure you want to continue connecting (yes/no)? **yes**
      Warning: Permanently added 'somehost' (RSA) to the list of known hosts.
  - All “known hosts” are stored in the ~/.ssh/known_hosts file
  - If it doesn’t match in the future – a warning will be generated

SSH Client Authentication

- Client proves its identity to the server with either option below
- Password
  - Typically default option
  - Client provides username/password which is checked against authorized users
- SSH Keys
  - Check if the client has the matching private key to the shared public key
  - Server encrypts a message with the shared public key
  - If the client can decrypt it, that is proof that the client has the matching private key
  - Client’s identity is verified

Lab Walkthrough

General Setup

- We will use lnxsrv09 as our “client”
  - Pretend like this is your own machine
  - By understanding how to configure this on Linux – you’ll get a better idea how to do the setup on your own Mac/Windows
- We will start with lnxsrv07 as the “server” we want to log in to

1. Log into lnxsrv09 and verify .ssh folder

- Go to your home directory – check for a .ssh directory.
- If is not there, create it

2. Generate SSH Keys

- What command can we use to generate SSH Keys?
- Remember to specify a password
- Which keys are generated?
- What permissions are set on the keys?

3. Update the ~/.ssh/authorized_keys file on lnxsrv07

- If the file doesn’t exist, create it
  - Note this file needs specific permissions
- Append the contents of your key file to the authorized_keys file
  - Which key should we append? The private or the public?

4. Confirm can login with SSH Key

- From lnxsrv09 you can use
  - ssh –i <relative_path_to_private_key> username@lnxsrv07.seas.ucla.edu

- Did you get any prompts?
  - If your key has a passphrase, you should get a prompt to use it

5. Make use of ssh-agent

- Typing in passwords sucks, so let’s use ssh-agent to facilitate
- Start ssh-agent, which command to use?
- Add our key to ssh-agent, which command to use?
- Now try logging into server, do you get a password?
  - ssh username@lnxsrv07.seas.ucla.edu

So what is ssh-agent

- ssh-agent is a daemon that holds key data and handles the signing of authentication data.
  - Part 3 of the lab (multi-hop ssh) shows a good way to make use of this

- NOTE: ssh-agent is a process that will end when your session ends, so it needs to be manually restarted everytime and each key added everytime
  - There are scripts that can be added to .bashrc to do the above automatically at the start of your session

6. Port-Forwarding with X-Window

Coming back to this at the end

7. Multi-Hop SSH

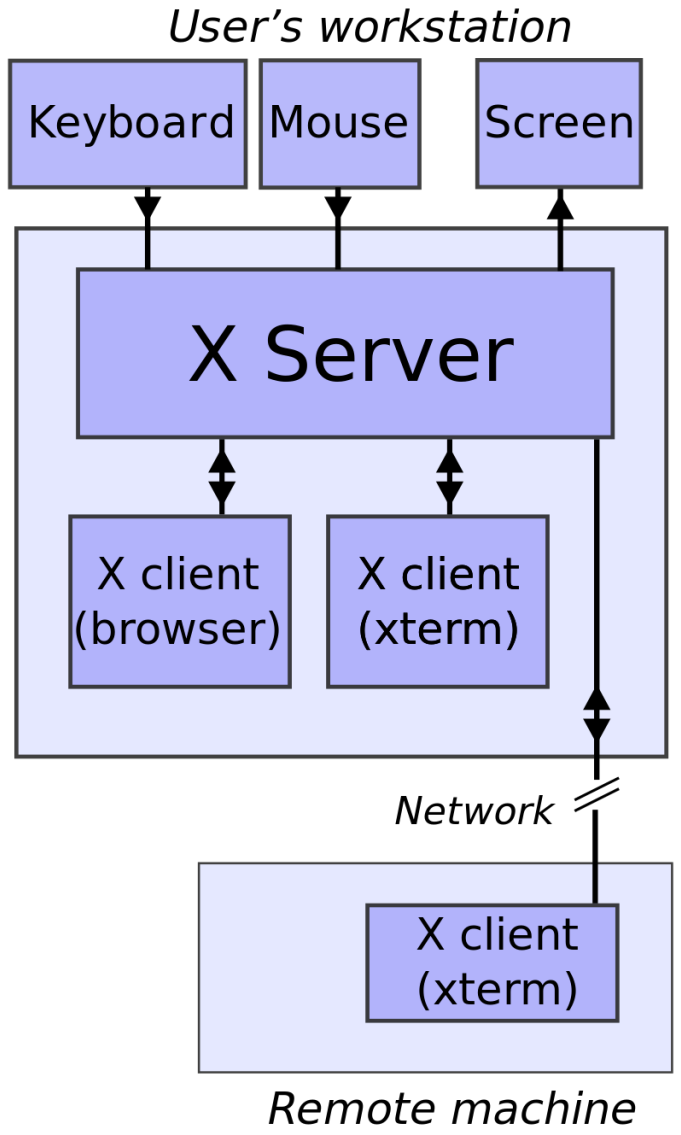
- Many real-world scenarios require you to connect to a “jump-box” and then you connect to the real computer you want.
- Possible solutions?
  - Can we install our private key on the jump-box?
  - Can we use ssh-agent to handle the sending of authentication data

- To test, try to sign in from lnxsrv09 to lnxsrv07 and then to lnxsrv10. What happens?
  - Check the link in the lab, does the -A option change the behavior?

6. Port-Forwarding with X-Window

- SSH can also support GUI connections! For example, running visual apps from a server
- This technique is called X-Forwarding to display information in X-Windows

- Use the SSH connection to send visual display information back to your machine



X-Forwarding Setup

- The setup below will use your own personal device and the server.
- Requirements
  - Install a X-Server on your machine: **XQuartz** for Mac or **Xming** on Windows
  - Enable X-Forwarding for your SSH Session
    - Either use the –X flag if you are directly using the command ssh
    - Windows users can enable the X-Forwarding Checkbox on Putty
- More setup details here - https://uit.stanford.edu/service/sharedcomputing/moreX