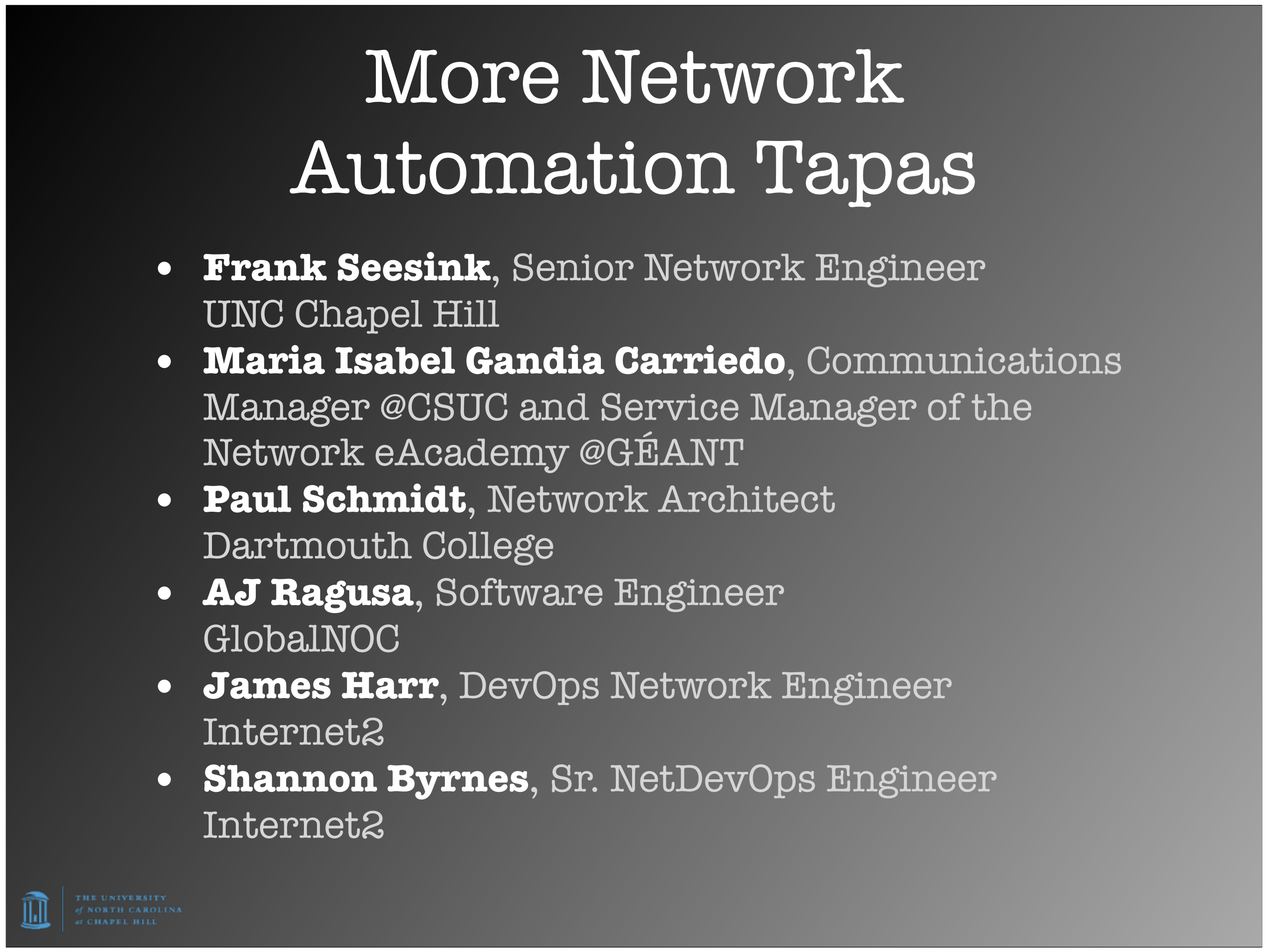
More Network Automation Tapas

Bite-sized talks to give the audience a little something to chew on

2023 INTERNET2 TECHNOLOGY exchange

THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL



More Network Automation Tapas

- **Frank Seesink**, Senior Network Engineer
  UNC Chapel Hill
- **Maria Isabel Gandia Carriedo**, Communications Manager @CSUC and Service Manager of the Network eAcademy @GÉANT
- **Paul Schmidt**, Network Architect
  Dartmouth College
- **AJ Ragusa**, Software Engineer
  GlobalNOC
- **James Harr**, DevOps Network Engineer
  Internet2
- **Shannon Byrnes**, Sr. NetDevOps Engineer
  Internet2

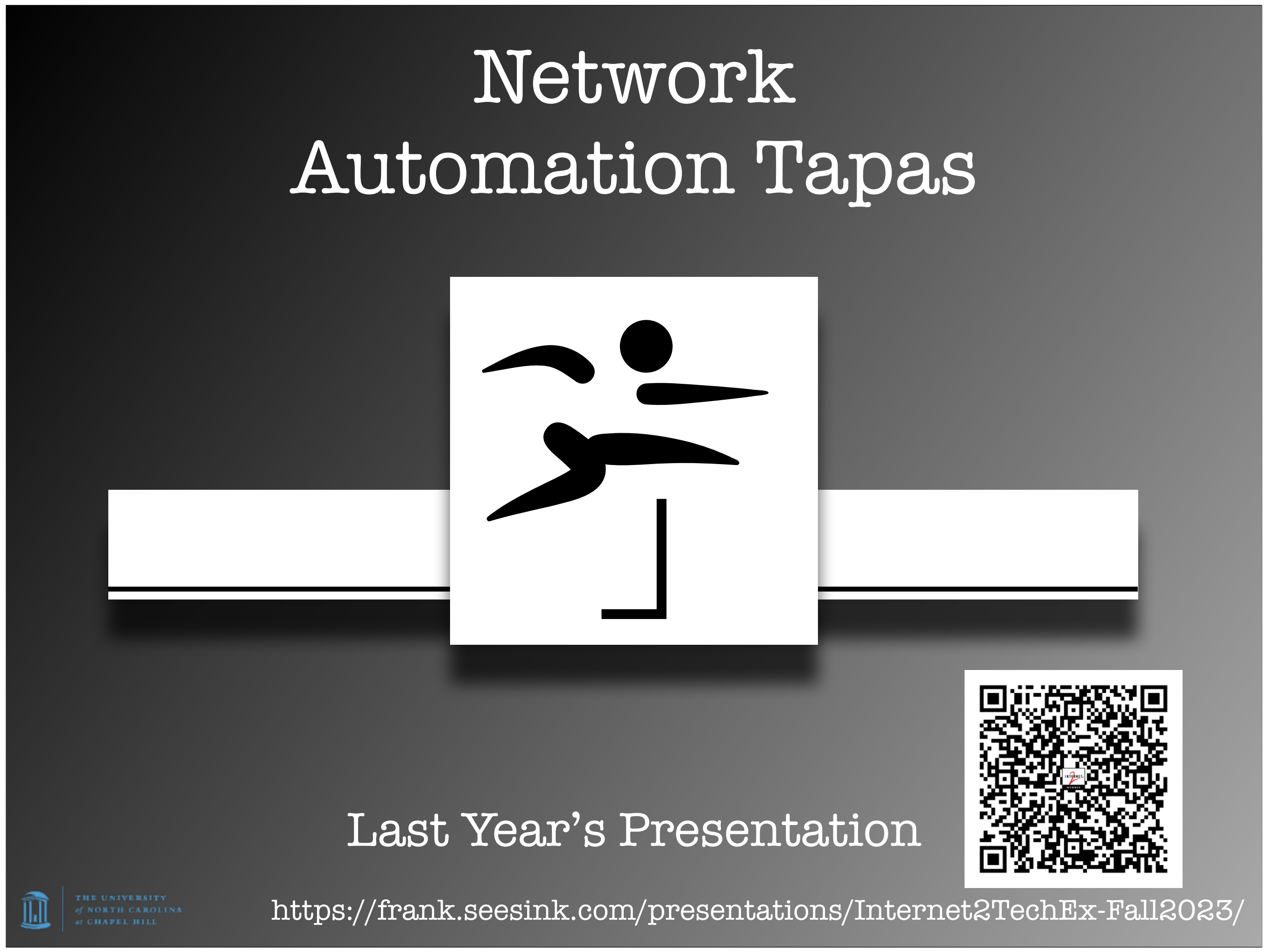
THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

Introduce members of panel

Why this session?

So why this session? Last year we introduced the notion of these small bite-sized presentations in response to one of the most common things I would hear from those already involved in network automation when asked how they would advise someone new to network automation. Overwhelmingly the response was "Just go do SOMETHING, ANYTHING, to get started." So the idea was to get folks over that first hurdle. (click)

This time, we continue that trend by presenting more topics in small chunks that hopefully help folks move forward in doing network automation.



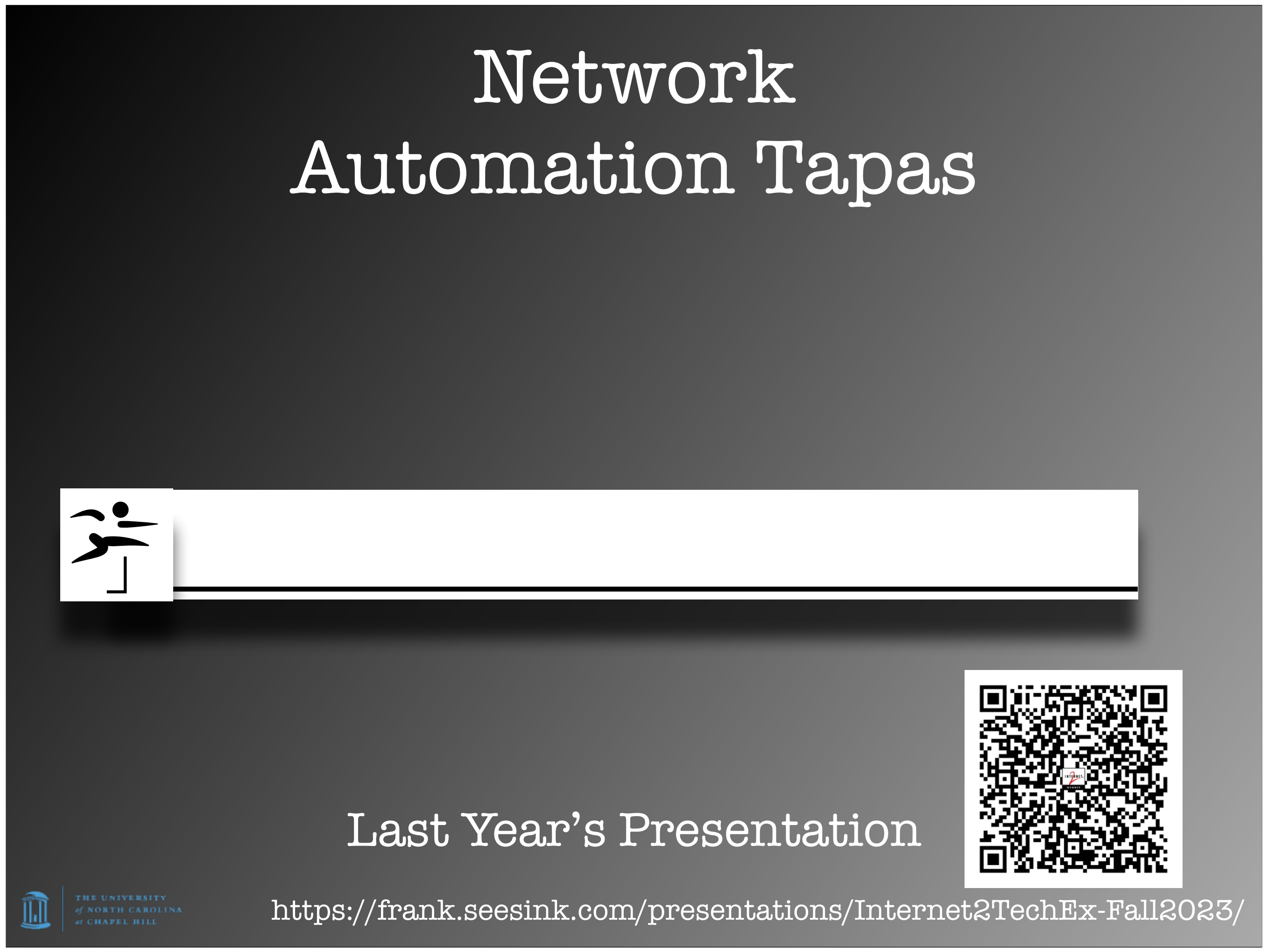
Network Automation Tapas

Last Year's Presentation

https://frank.seesink.com/presentations/Internet2TechEx-Fall2023/

So why this session? Last year we introduced the notion of these small bite-sized presentations in response to one of the most common things I would hear from those already involved in network automation when asked how they would advise someone new to network automation. Overwhelmingly the response was "Just go do SOMETHING, ANYTHING, to get started." So the idea was to get folks over that first hurdle. (CLICK)

This time, we continue that trend by presenting more topics in small chunks that hopefully help folks move forward in doing network automation.



Network Automation Tapas

Last Year's Presentation

https://frank.seesink.com/presentations/Internet2TechEx-Fall2023/

So why this session? Last year we introduced the notion of these small bite-sized presentations in response to one of the most common things I would hear from those already involved in network automation when asked how they would advise someone new to network automation. Overwhelmingly the response was "Just go do SOMETHING, ANYTHING, to get started." So the idea was to get folks over that first hurdle. (CLICK)

This time, we continue that trend by presenting more topics in small chunks that hopefully help folks move forward in doing network automation.

“A tapa (Spanish pronunciation: ['tapa]) is an appetizer or snack in Spanish cuisine. Tapas can be combined to make a full meal, and can be cold (such as mixed olives and cheese) or hot (such as chopitos, which are battered, fried baby squid, or patatas bravas).”
From https://en.wikipedia.org/wiki/Tapas

Much as tapas are a series of snacks, the topics presented here are intended just to give a few tastes from the network automation space. That is, to provide you with a place to get started or something new to chew on.

PKI in 5 Minutes

What IS PKI??

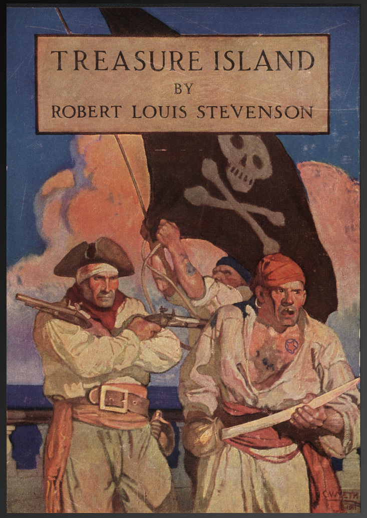
"A public key infrastructure (PKI) is a set of roles, policies, hardware, software and procedures needed to create, manage, distribute, use, store and revoke digital certificates and manage public-key encryption."

— https://en.wikipedia.org/wiki/Public_key_infrastructure

BAH! Too complicated! Let’s simplify.

THE UNIVERSITY *of* NORTH CAROLINA *at* CHAPEL HILL

Pirates



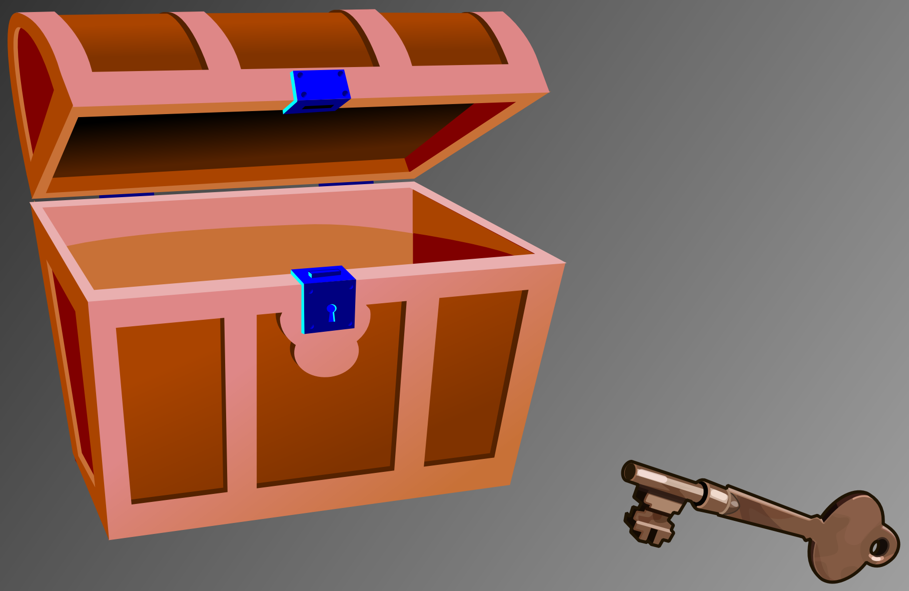
Who here has ever read a book on or watched a movie about pirates? What do most of those stories involve? TREASURE!

And to store that treasure, you need... a TREASURE CHEST!

Treasure Chest

Let’s focus on that treasure chest.

Typical Treasure Chest

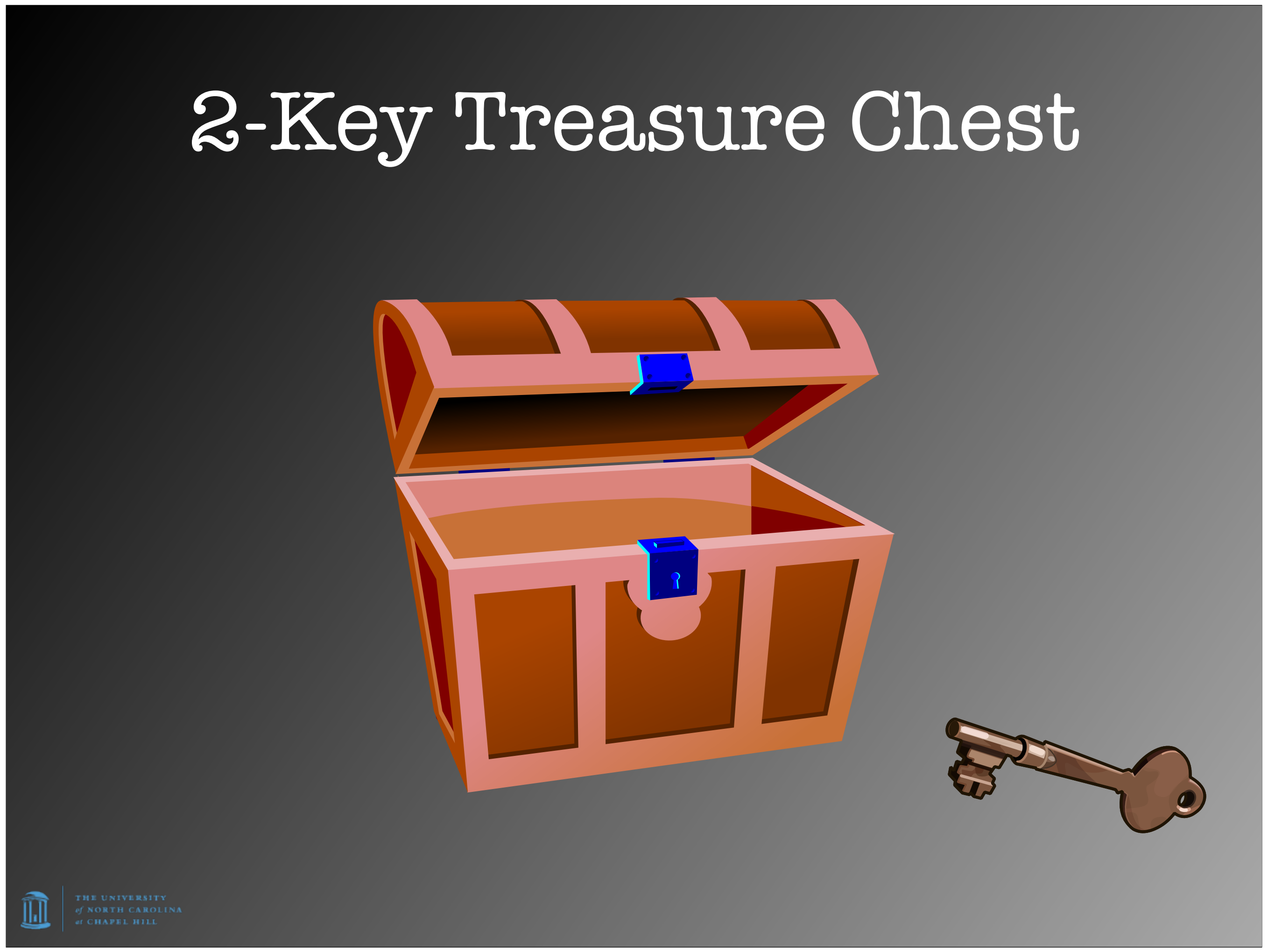


Now for a treasure chest to be useful, it needs a lock.
And that lock requires a key.
Put a key in the lock, turn it, and your treasure is now safe.

Typical Treasure Chest

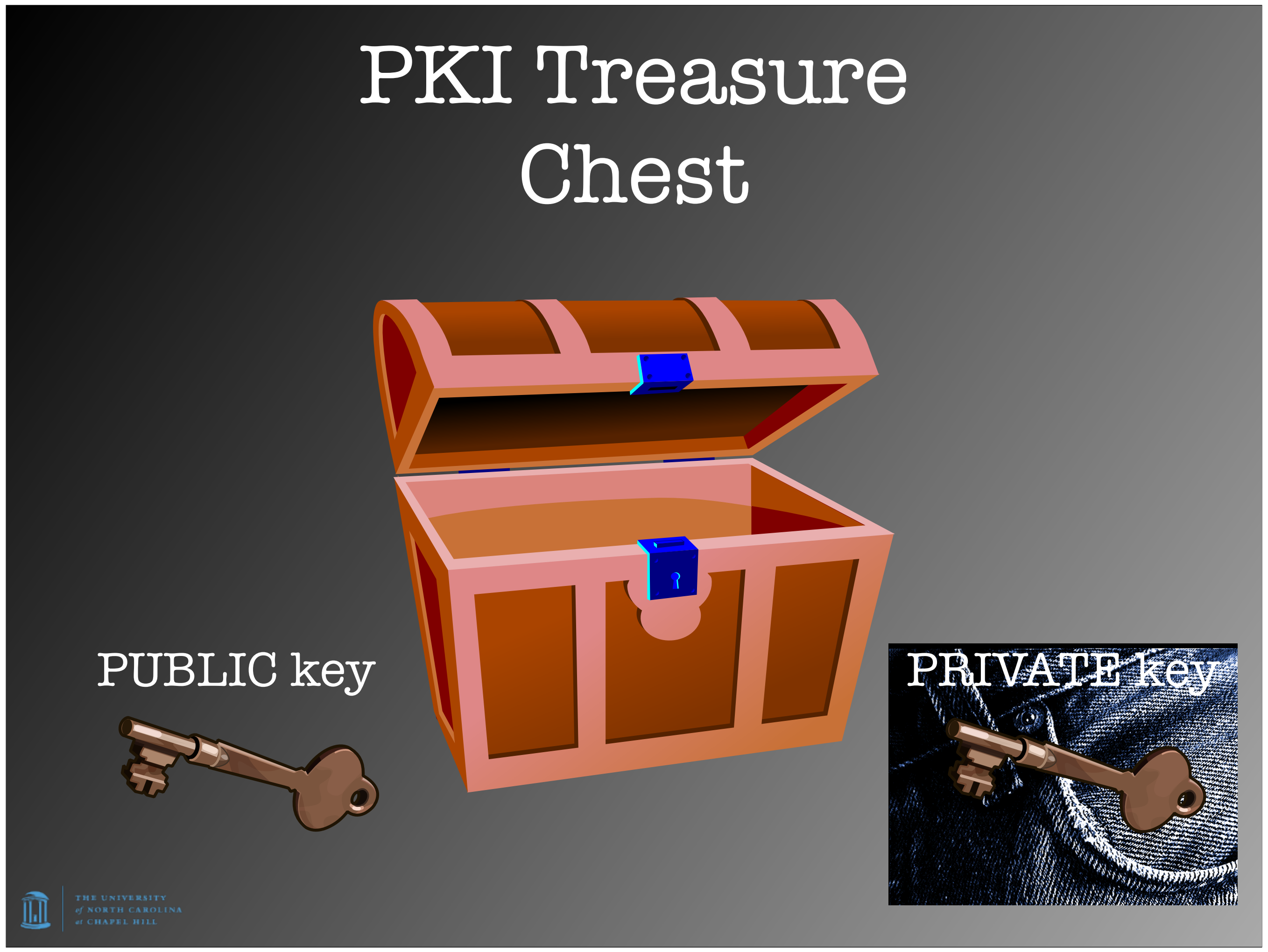
THE UNIVERSITY
of NORTH CAROLINA
at CHAPEL HILL

With me so far?



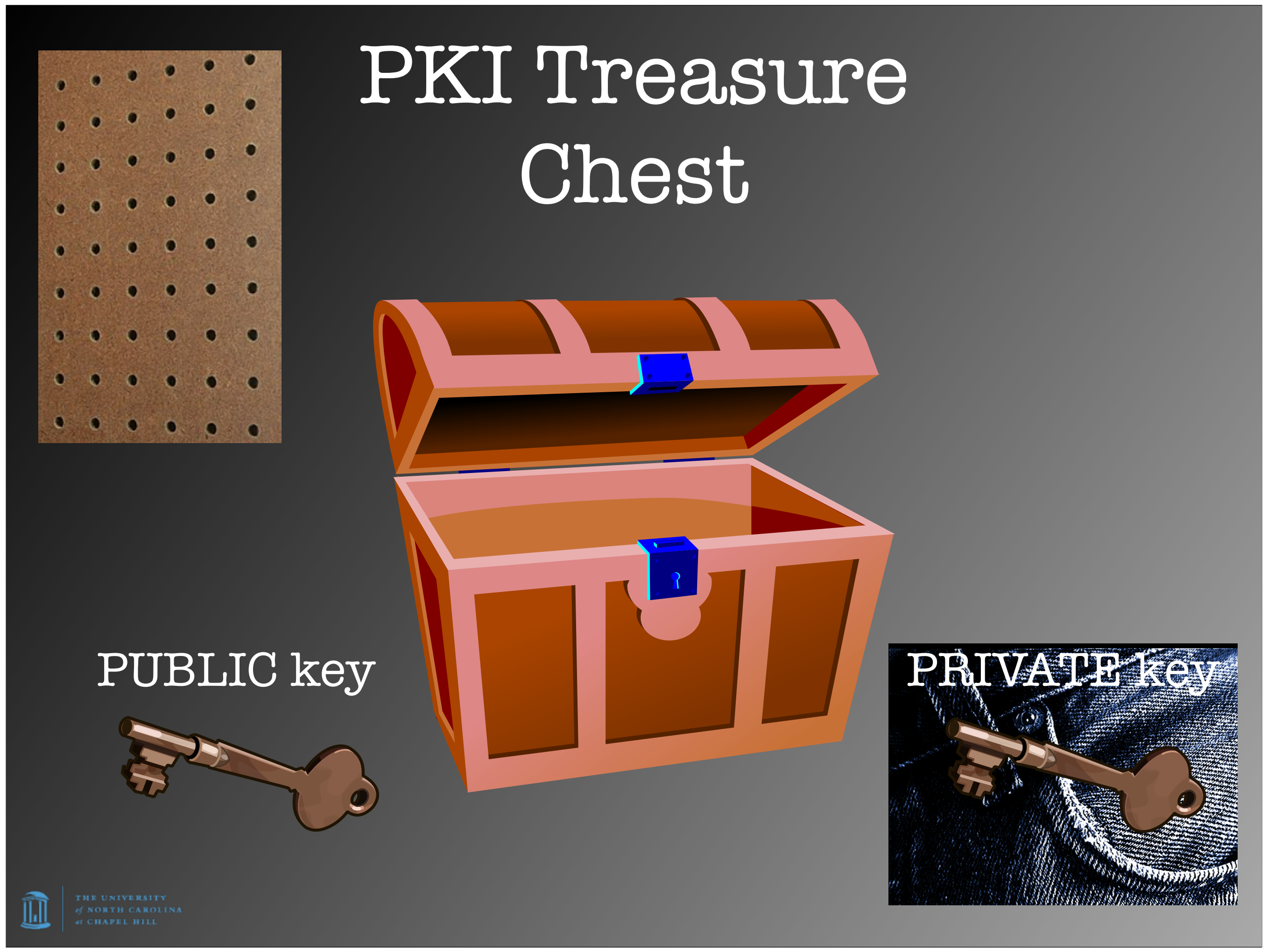
2-Key Treasure Chest

Alright, let's imagine this is a special treasure chest. It requires not ONE (1) but TWO (2) keys.
If you lock the chest with the key on the left, only the key on the right can open it.
If you lock the chest with the key on the right, only the key on the left can open it.
If you understand this, you now understand the fundamental building block of PKI. But let's continue.



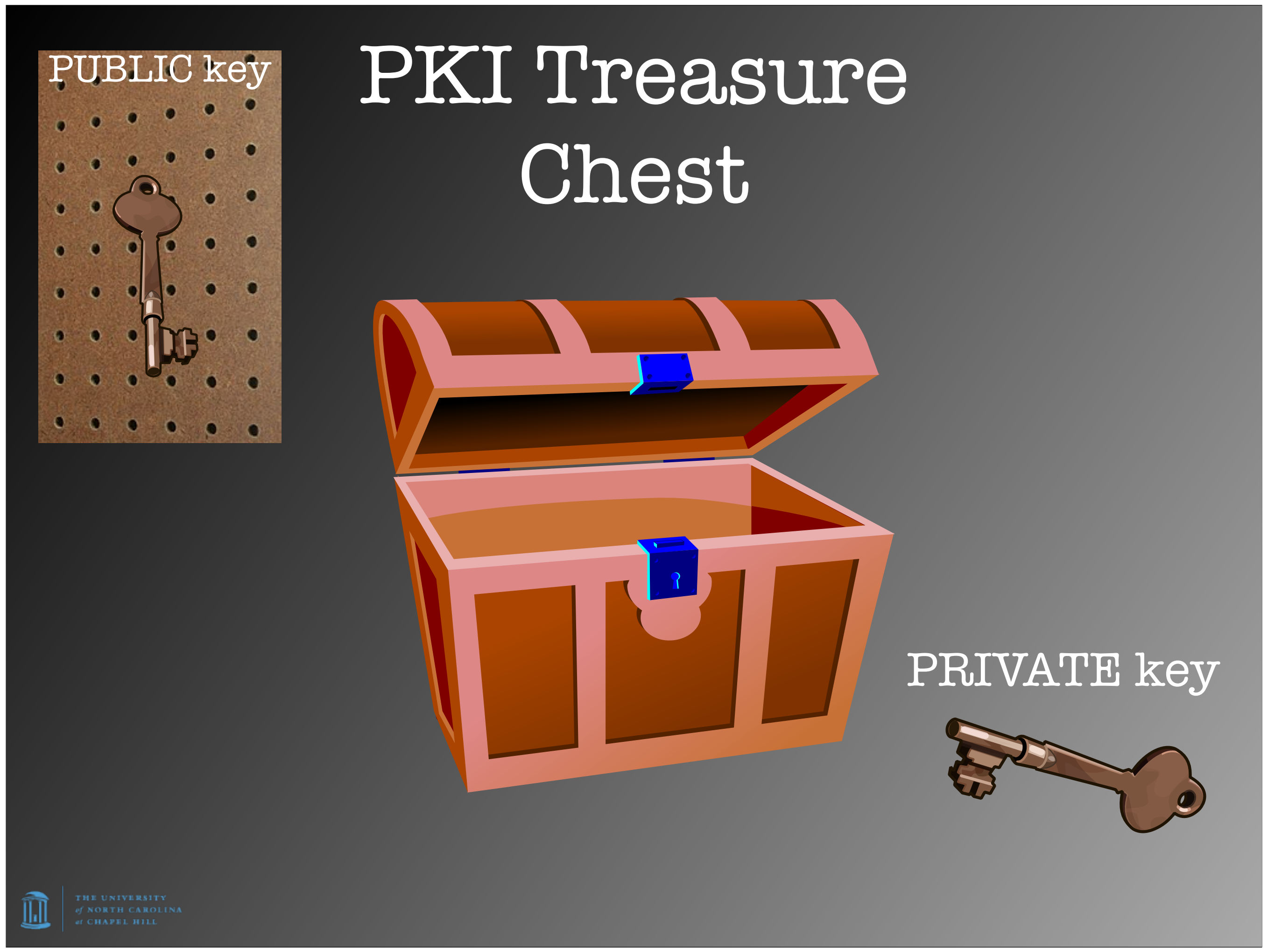
PKI Treasure Chest

We name these keys the PRIVATE key and the PUBLIC key. Hence the “Public Key” in Public Key Infrastructure (PKI).
The PRIVATE key you keep in your pocket. You never share it or otherwise let others have access to it. Hence “private”.



PKI Treasure Chest

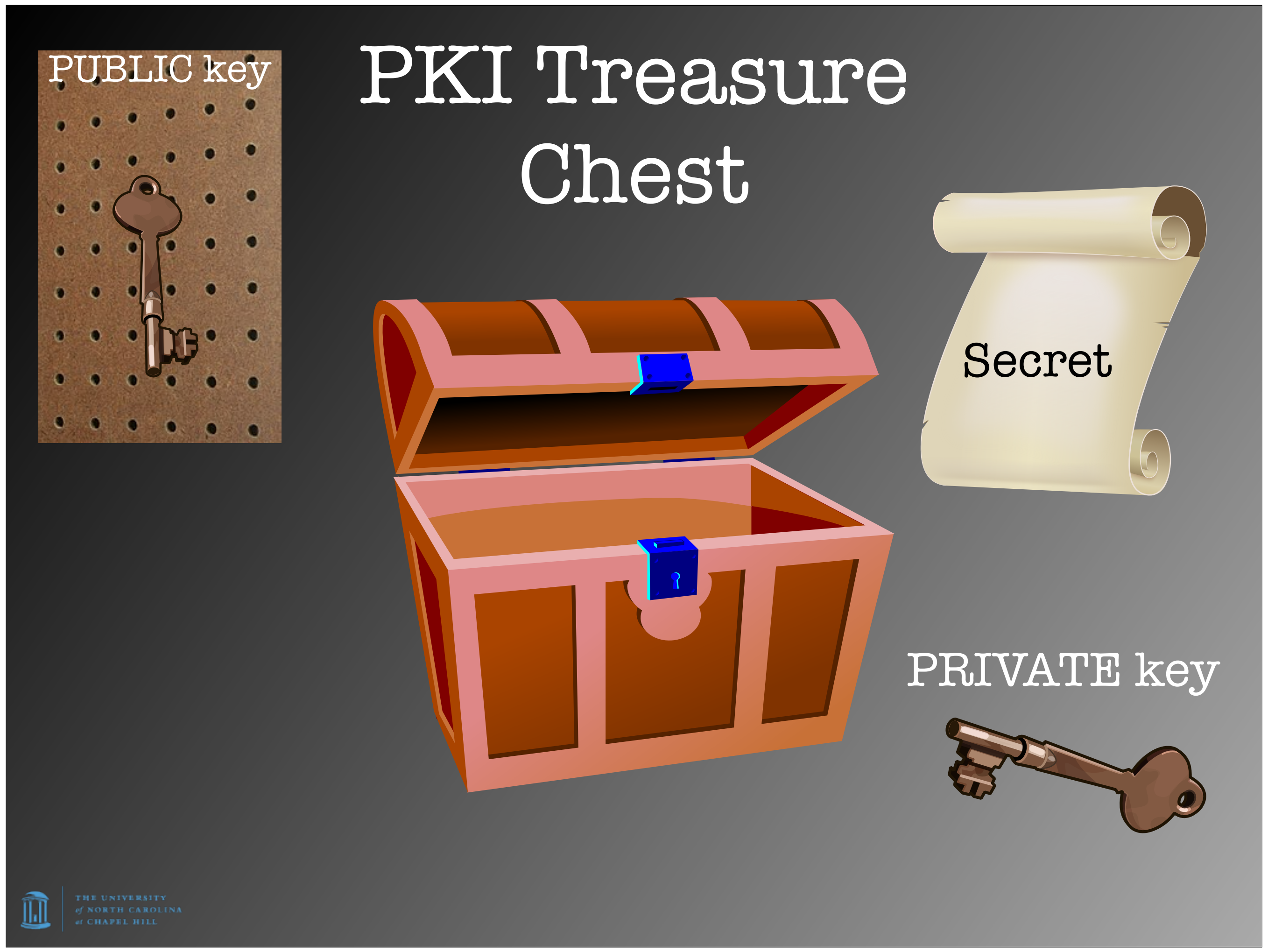
The PUBLIC key is just that: public. So you put that out in the world where anyone can get to it.



PKI Treasure Chest

THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

Now let's walk through a basic example.



PKI Treasure Chest

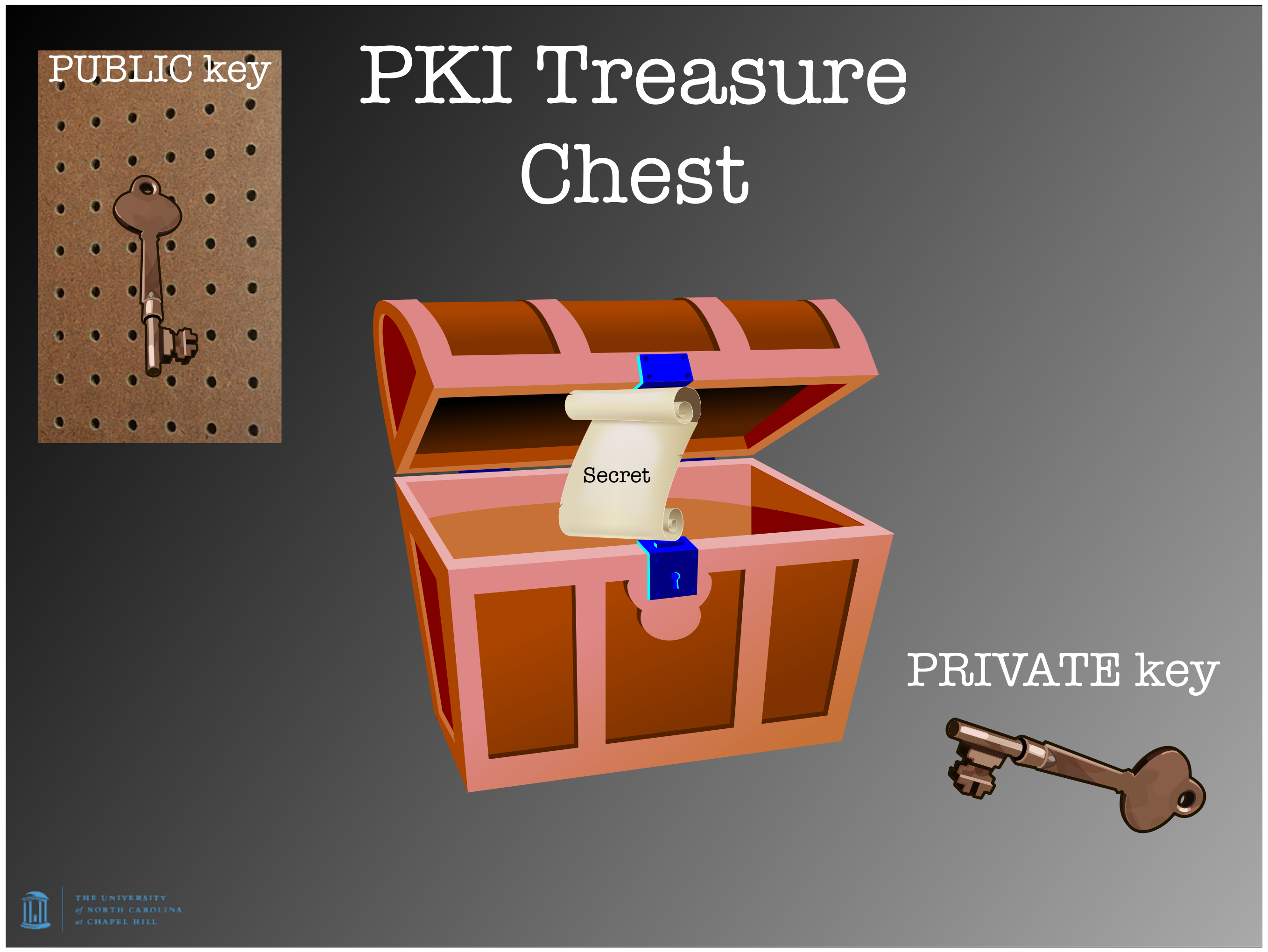
We have something we wish to protect. So we place it in the treasure chest and we lock it with the PRIVATE key.

PKI Treasure Chest

PUBLIC key
Locked with PRIVATE key
PRIVATE key

The secret is now secure and we can send it. And because we locked it with the PRIVATE key, the ONLY key out in the world that can now unlock the chest is the PUBLIC key.

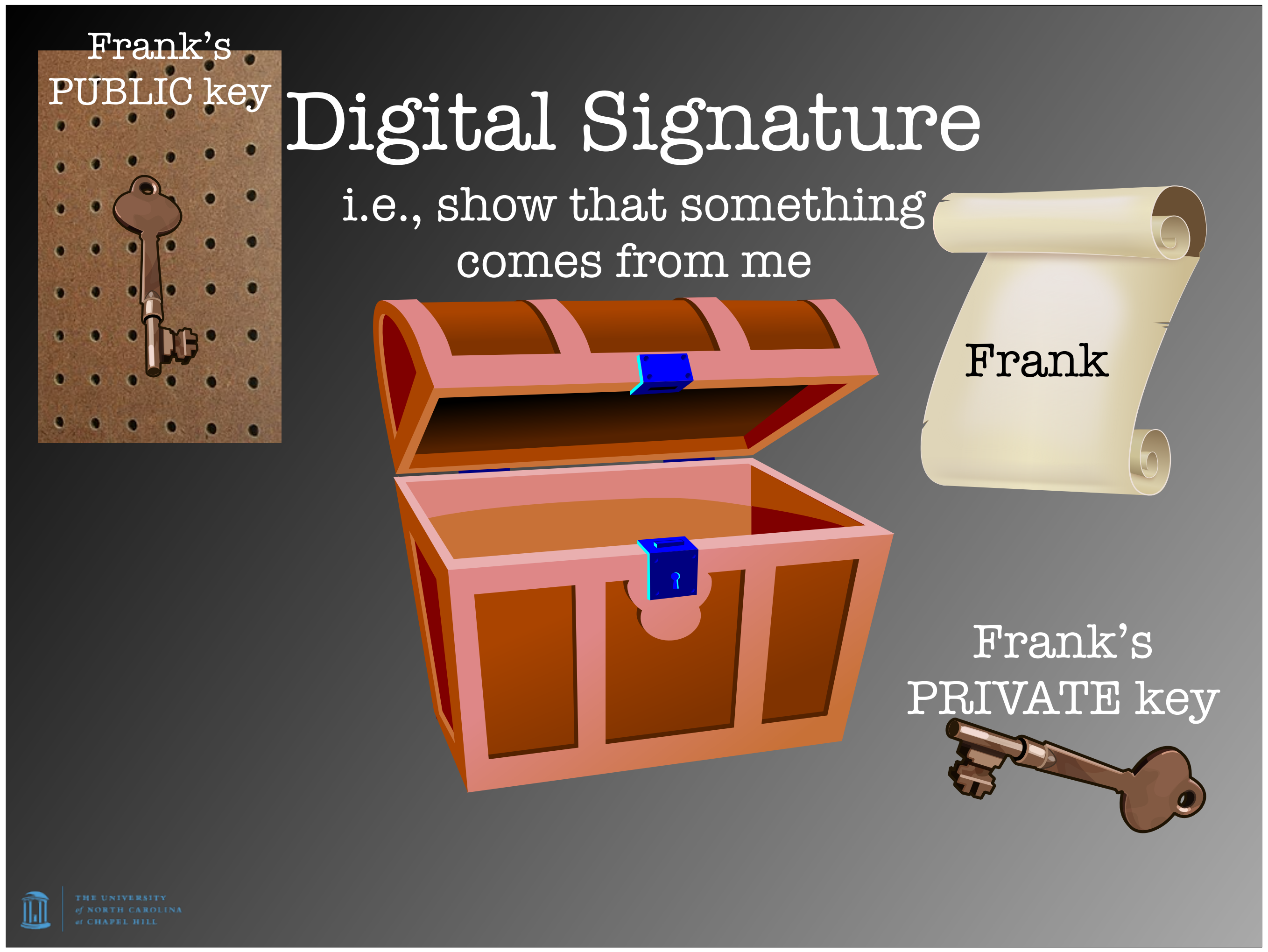
So on the other end, they then go to the peg board, take the PUBLIC key, and unlock the treasure chest.



PKI Treasure Chest

They now have the information that was secured.

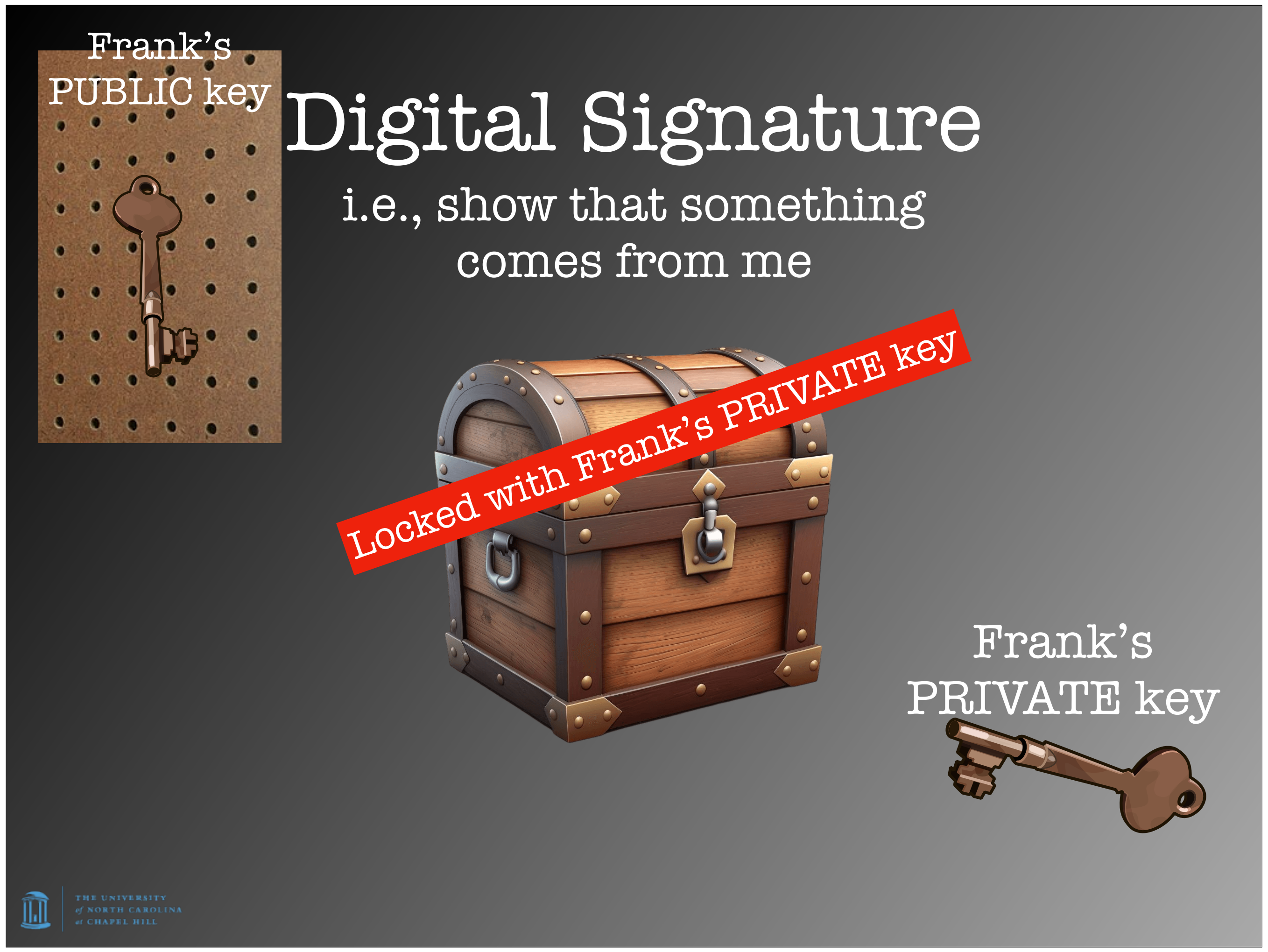
So How Is This PKI?



Digital Signature

i.e., show that something comes from me

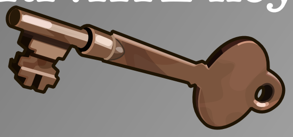
Let's say I want to send you a message. And I want you to be sure that the message came from me and no one else. This is known as a digital signature. So I take my message, put it in a treasure chest, and I lock it with my PRIVATE key.



Digital Signature

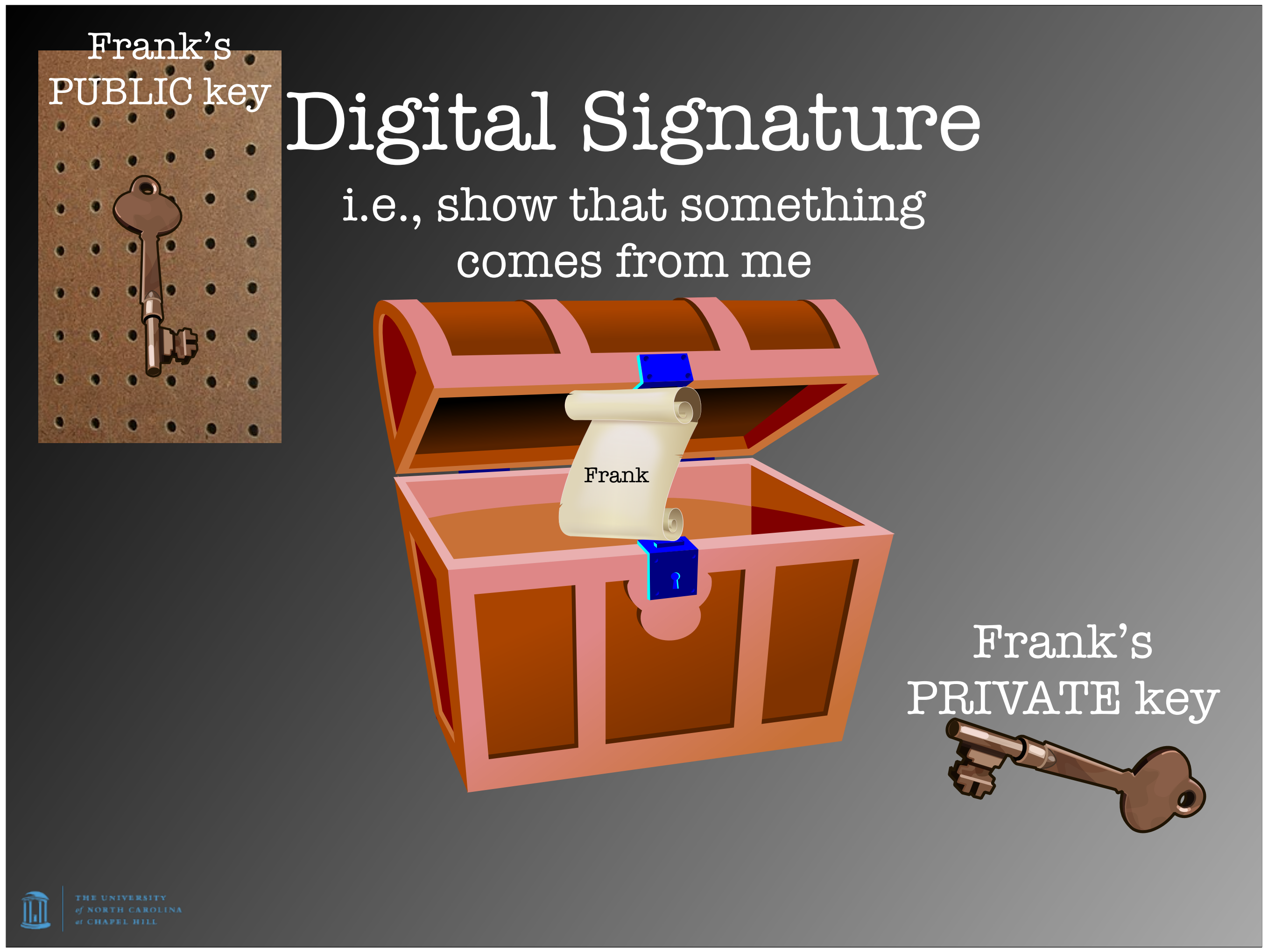
i.e., show that something comes from me

Locked with Frank's PRIVATE key



This treasure chest is marked as being locked with Frank's PRIVATE key.
I then send it to you.

When you get it, you then take my PUBLIC key off the peg board, and you use that to unlock the chest.

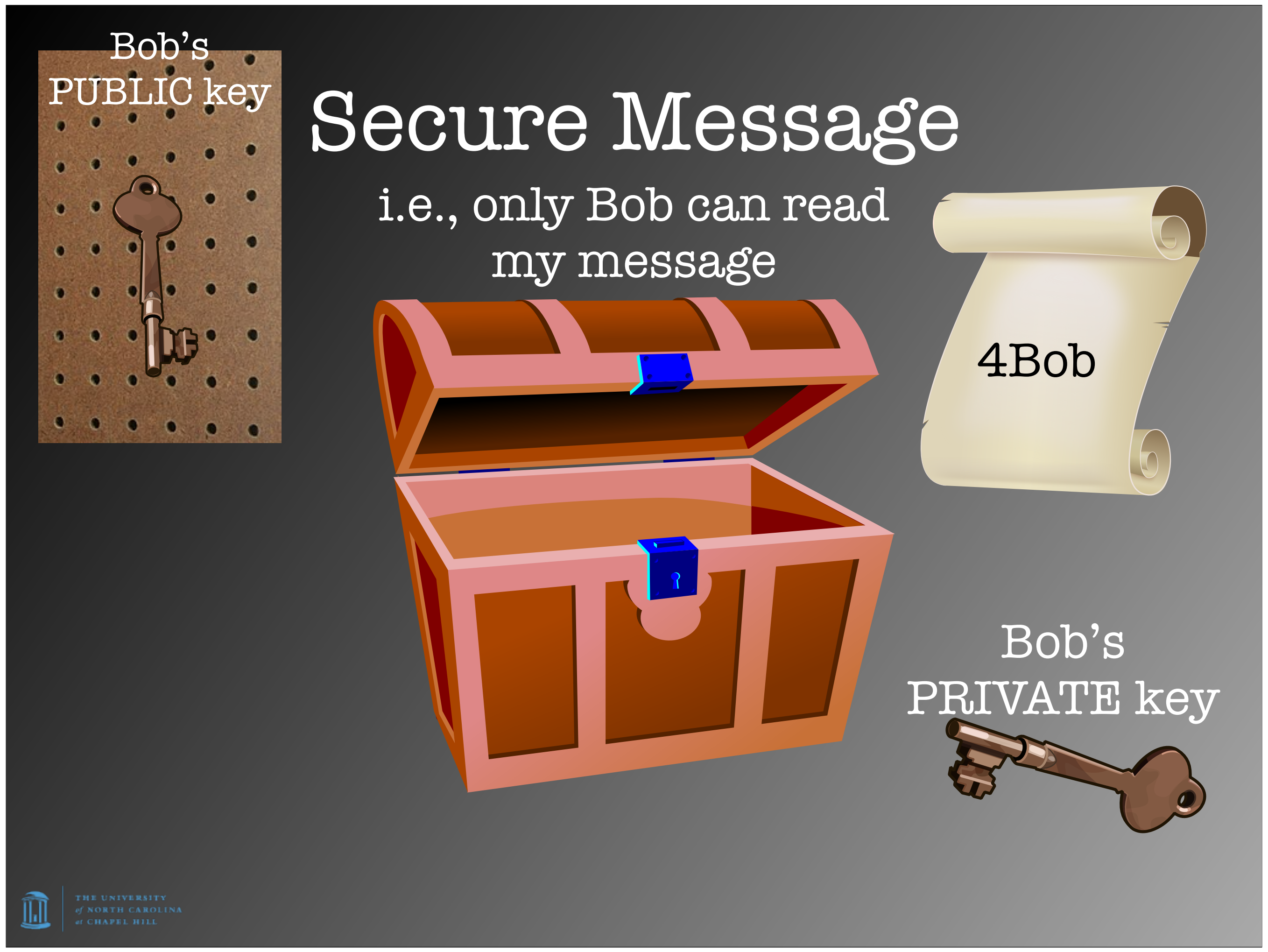


Digital Signature

i.e., show that something comes from me

You now retrieve the message I sent. And you can be sure that it came from me, because my PUBLIC key unlocked the chest. And the only person who has the matching PRIVATE key is me.

However, this message could be seen by anyone since all it takes is access to my PUBLIC key.



Secure Message

i.e., only Bob can read my message

So now let's say I wish to send a message, but I ONLY want Bob to be able to read it.
So I put my message in the chest, but THIS time I lock it with Bob's PUBLIC key.

Secure Message

i.e., only Bob can read my message

Bob's
PUBLIC key

Locked with Bob's PUBLIC key

This treasure chest is marked as being locked with Bob's PUBLIC key.
I then send it to Bob. If anyone else intercepts it, I know that it is safe, as the ONLY person who can unlock this chest is Bob, since it requires his PRIVATE key.
So Bob gets it, sees how it was locked, and he takes out his PRIVATE key and unlocks the chest.

Secure Message

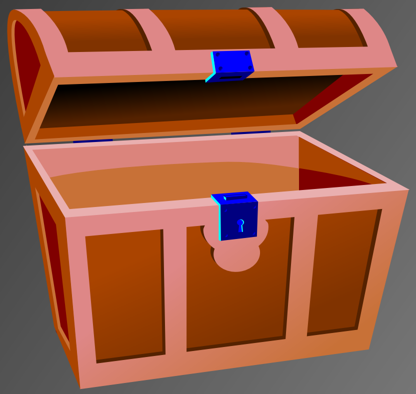
i.e., only Bob can read my message

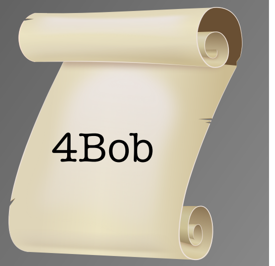
Bob's
PUBLIC key
4Bob
Bob's
PRIVATE key

Bob now KNOWS that this message was intended for him and him alone.
HOWEVER, it could have been sent to Bob by ANYONE.
So that bring us to…

End-to-End Encryption

PUBLIC
Keys
Bob
Frank

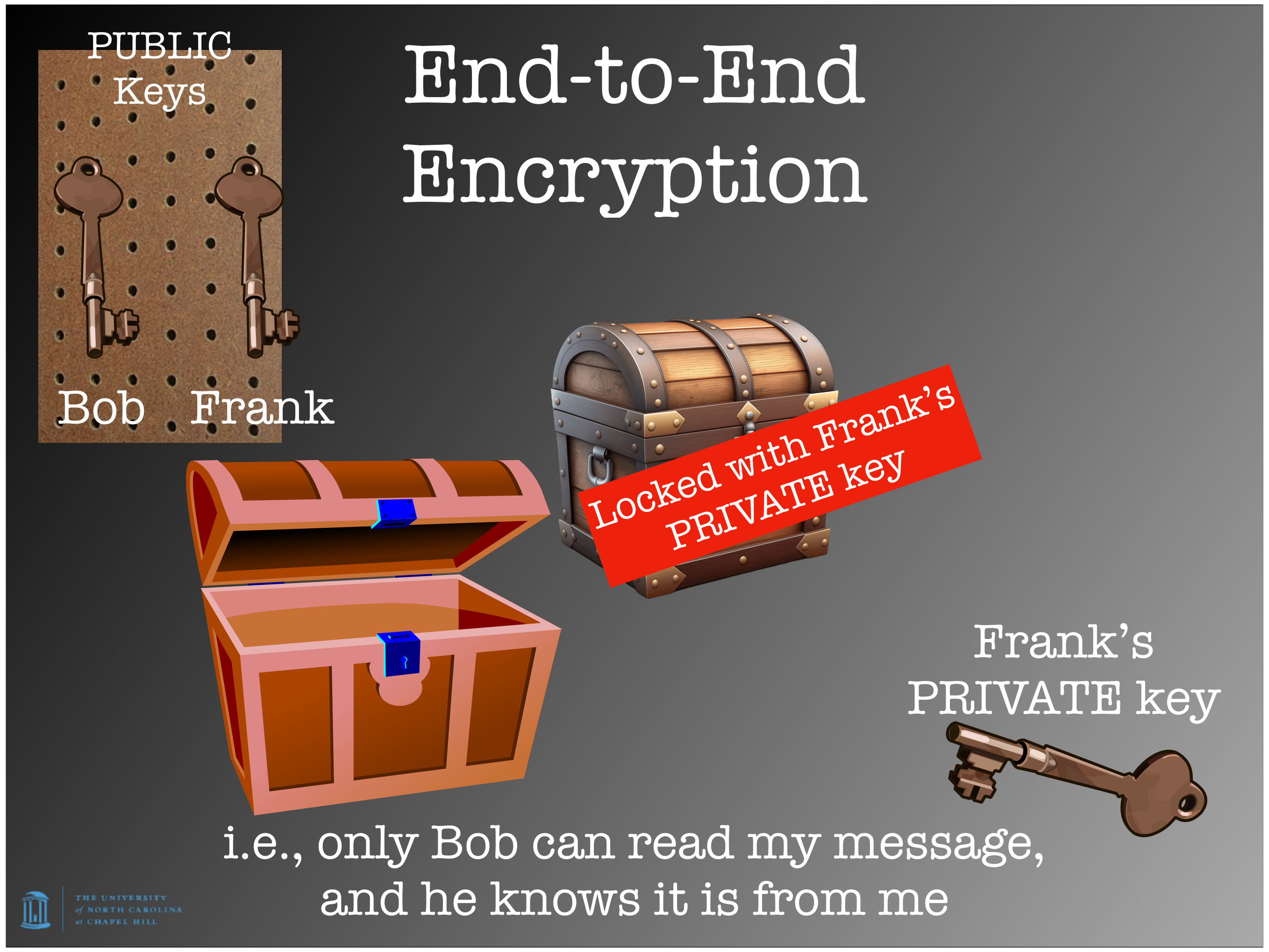




i.e., only Bob can read my message,
and he knows it is from me

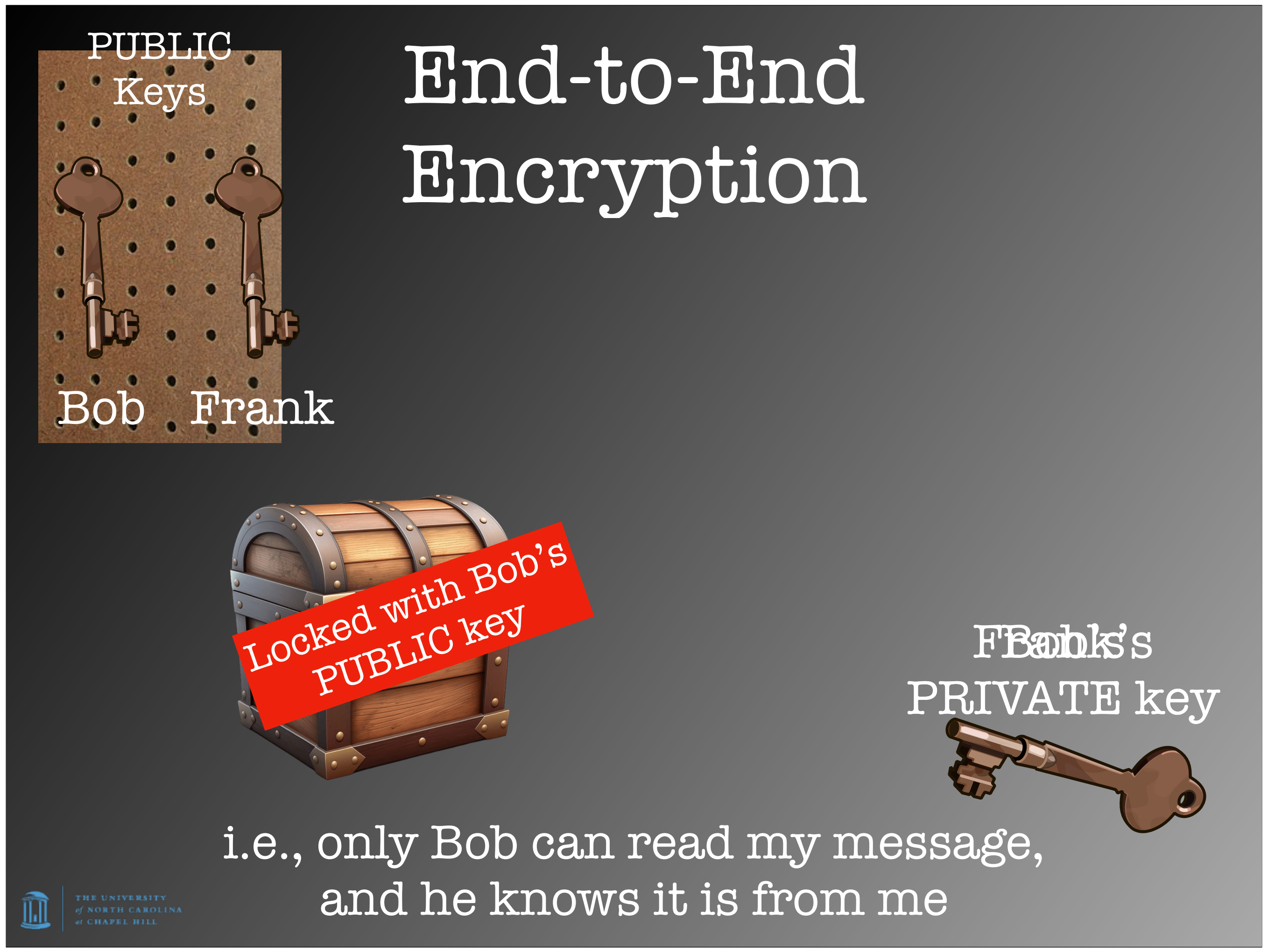
End-to-End (e2e) encryption.
This time, in order to make sure that Bob knows that it was ME that sent the message, and for me to know that ONLY Bob can read it, I will combine the last 2 things together.
That is, first I will put my message into a chest and lock it with my PRIVATE key.

End-to-End Encryption



i.e., only Bob can read my message, and he knows it is from me

This treasure chest is marked as being locked with Frank's PRIVATE key.
Next I will place this chest inside ANOTHER chest, only THIS time I will lock the outside chest with Bob's PUBLIC key.



PUBLIC
Keys

Bob Frank

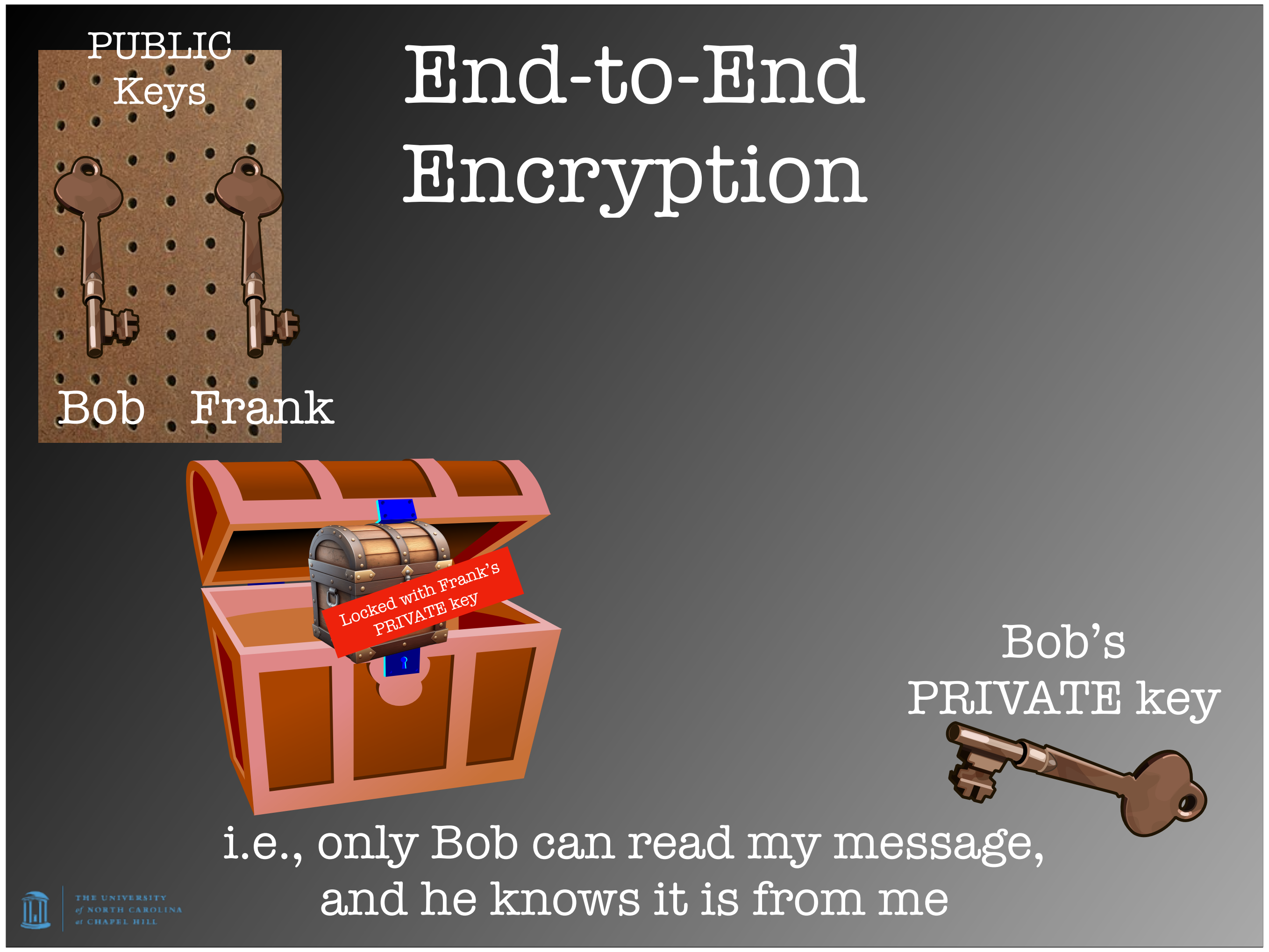
End-to-End Encryption

Locked with Bob's
PUBLIC key

Frank's
Bob's
PRIVATE key

i.e., only Bob can read my message,
and he knows it is from me

At this point we have a treasure chest that is marked as being locked with Bob's PUBLIC key.
I then send it to Bob.
Again, it cannot be read by anyone else. And when Bob receives it, he uses his PRIVATE key to unlock the chest.

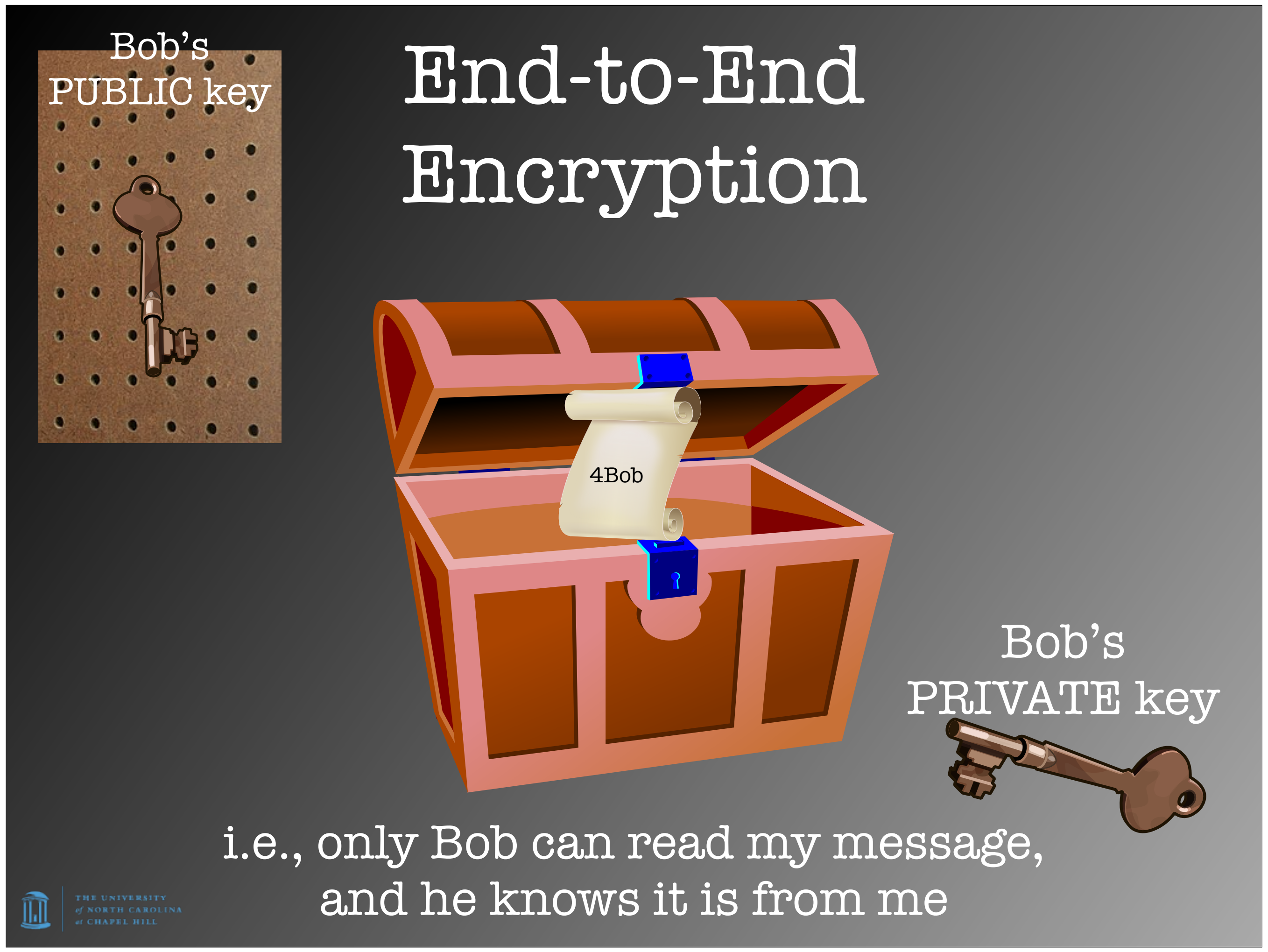


End-to-End Encryption

PUBLIC Keys
Bob
Frank
Locked with Frank's PRIVATE key
Bob's PRIVATE key

i.e., only Bob can read my message, and he knows it is from me

Inside that chest, Bob finds another chest, this one marked as being locked with Frank's PRIVATE key.
So Bob goes over to the public peg board, finds my PUBLIC key, and unlocks that chest using it.

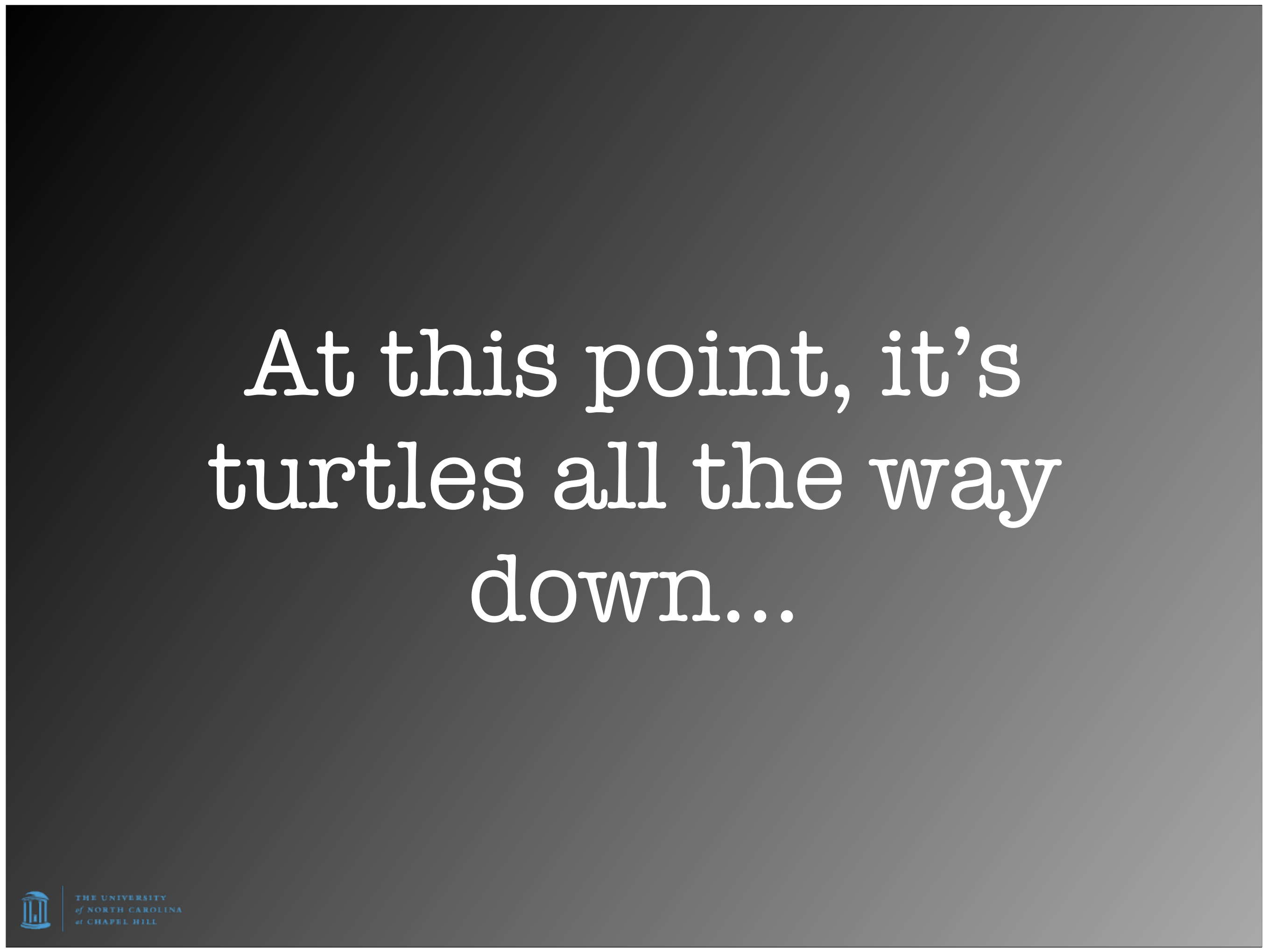


End-to-End Encryption

Bob's PUBLIC key
4Bob
Bob's PRIVATE key

i.e., only Bob can read my message, and he knows it is from me

At this point Bob gets my message. He knows it was intended for him and only him. And he knows that it came from me. If he wants to reply, he simply follows similar steps, first locking a message with his PRIVATE key, and then locking that in another chest using my PUBLIC key.



At this point, it's turtles all the way down…

THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

With this basic understanding of the fundamental building block of PKI, you can use this analogy to make sense of almost everything PKI related.

e.g., Certificate Authorities (CAs)

For example, Certificate Authorities (CAs) are nothing more than an extension of this treasure chest analogy. Certificates (such as SSL/TLS certificates) are just documents which contain information such as FQDNs, expiration dates, etc. And CAs (such as Verisign and Comodo) are just servers which digitally sign those certificates with their private keys (i.e., place the certificate in a chest and lock with their PRIVATE key), where the CA's public keys are stored on client devices in a trusted manner (such as a web browser).

To use our analogy, what would stop someone from replacing my PUBLIC key on the peg board with their own PUBLIC key, just labeling it to claim it was "Frank's PUBLIC key"? Well nothing. And if they did this, they could then use their own PRIVATE key to secure a message and send it to others, labeling it as being secured with my PRIVATE key. These folks could thereby be tricked into believing a message was sent from me when it was not.

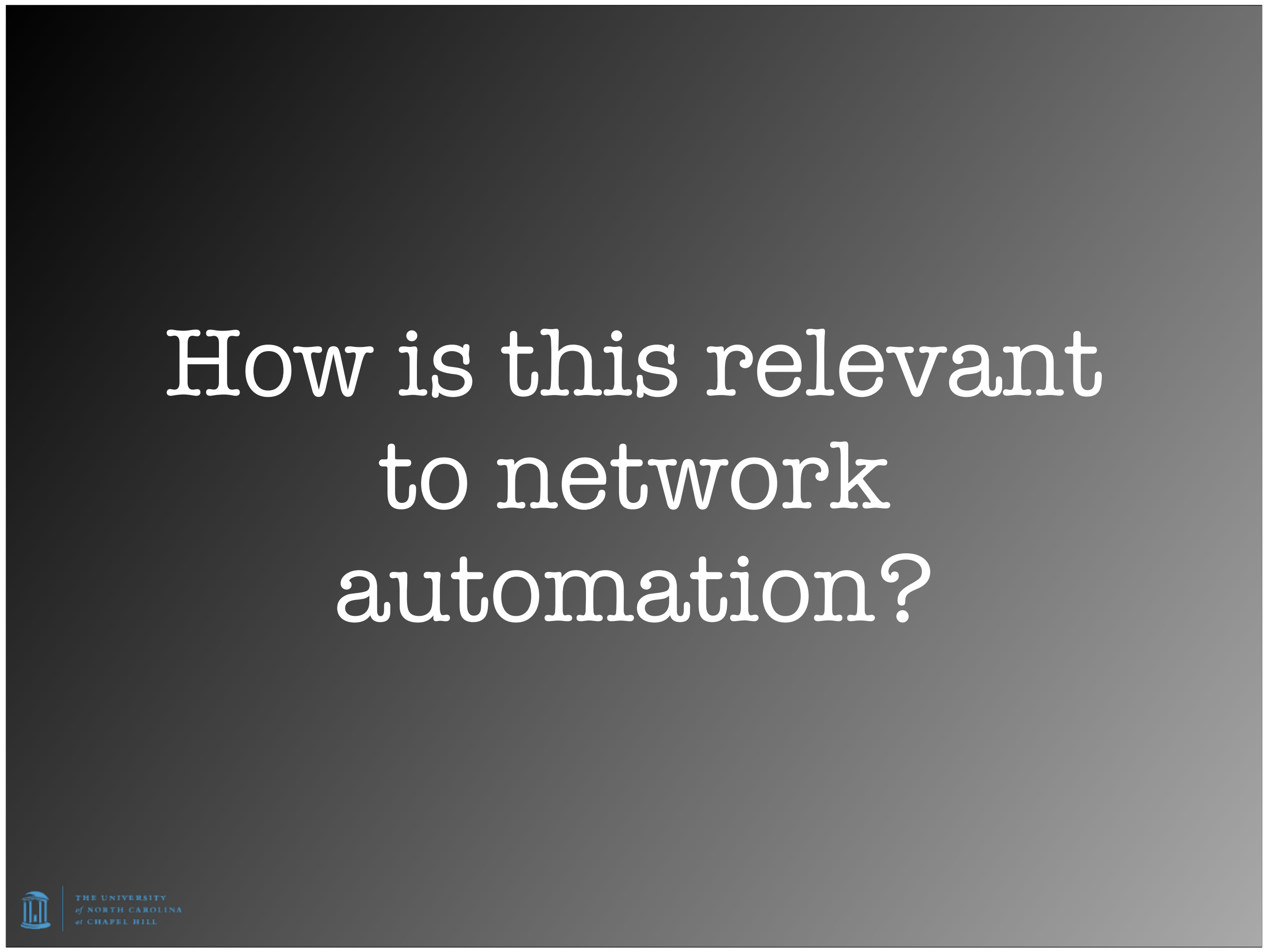
This is where "trust anchors" come into play. Everything with encryption relies upon trust in one form or another. But what does that mean here?

Well, in the case of, say, SSL/TLS certificates, when a web server presents your browser with a certificate claiming to be "www.google.com", how does the browser know whether the web server is lying? This is where the CAs come in. When someone like Google creates an SSL/TLS certificate for their website, they first generate a CSR (Certificate Signing Request), which is exactly as its name implies. The process involves Google generating its own PUBLIC/PRIVATE key pair, then sending the PUBLIC key along with the CSR over to a CA such as Verisign to digitally sign. That is, the CA then encrypts Google's PUBLIC key with Verisign's PRIVATE key (i.e., they put Google's PUBLIC key in a chest and lock it with Verisign's PRIVATE key, and label the chest as such). This is what the Google web server then presents to your browser when you visit their HTTPS site.

Now your browser has stored within it a set of CA PUBLIC keys. These were provided by the various CAs to each browser vendor in a trusted manner. (Maybe someone from each CA drove over to each browser maker and hand delivered them. The point being this is the trust anchor portion.)

So when you visit www.google.com and the web server sends you its SSL/TLS certificate (which lists which CA PUBLIC key locked its chest), the first thing your browser does is look up the matching CA's PUBLIC key and decrypts the certificate (unlocks the chest). If it decrypts (if the chest unlocks), then the browser knows that the CA signed the certificate and that the browser can trust it.

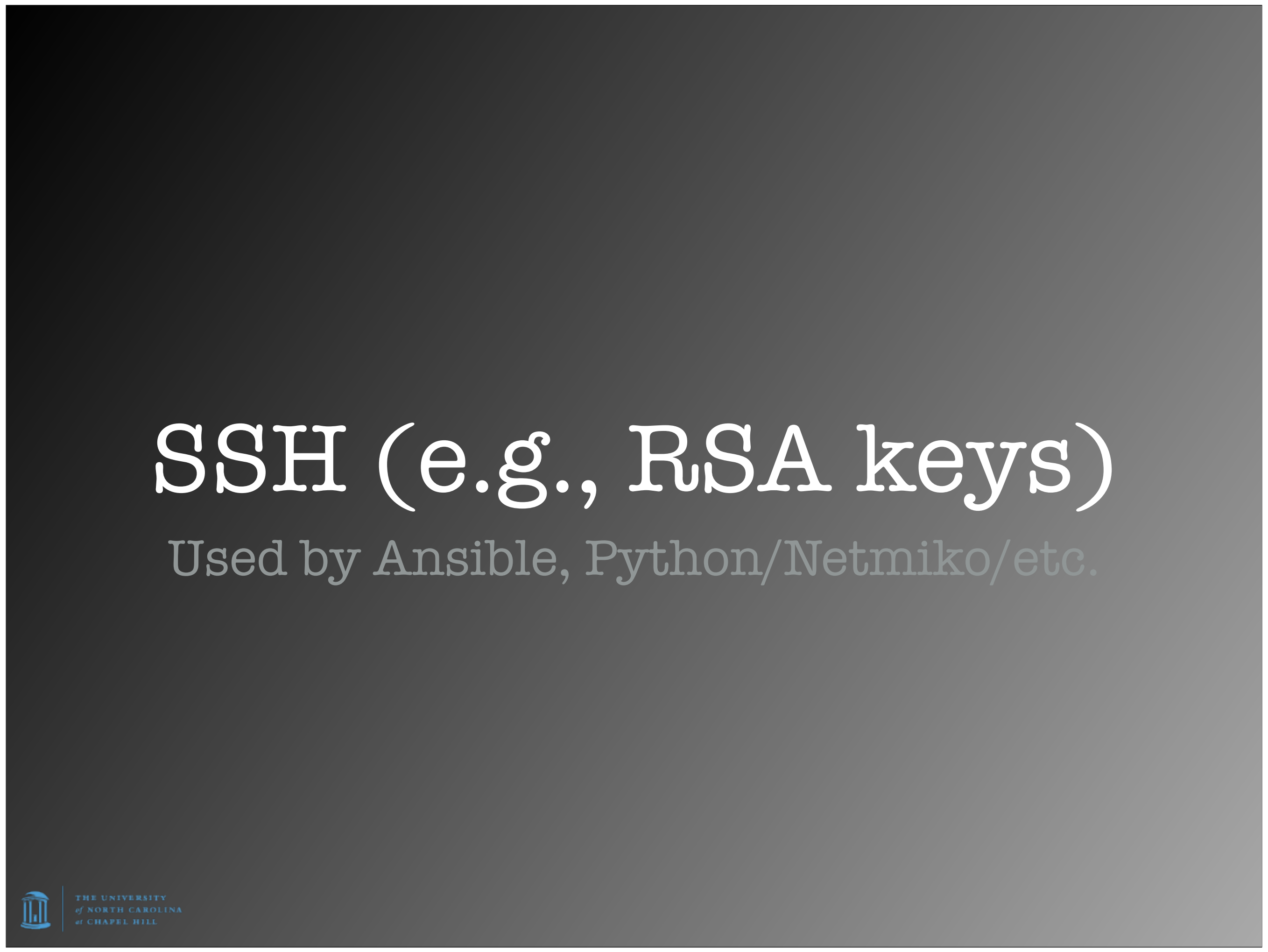
If anyone other than Google tries to submit a CSR for “www.google.com", the CA should prevent that. It is the CA’s duty to perform due diligence in this matter. In fact, if a CA is found NOT to perform its duties, it can be removed from the list of trusted CAs that the browser uses. If that happens, any and ALL certificates issued by that CA will be rejected. This happened recently when Google removed Entrust from its Chrome browsers as did Mozilla from Firefox. (Example article: https://www.theregister.com/2024/06/28/google_axes_entrust_over_six/)



How is this relevant to network automation?

THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

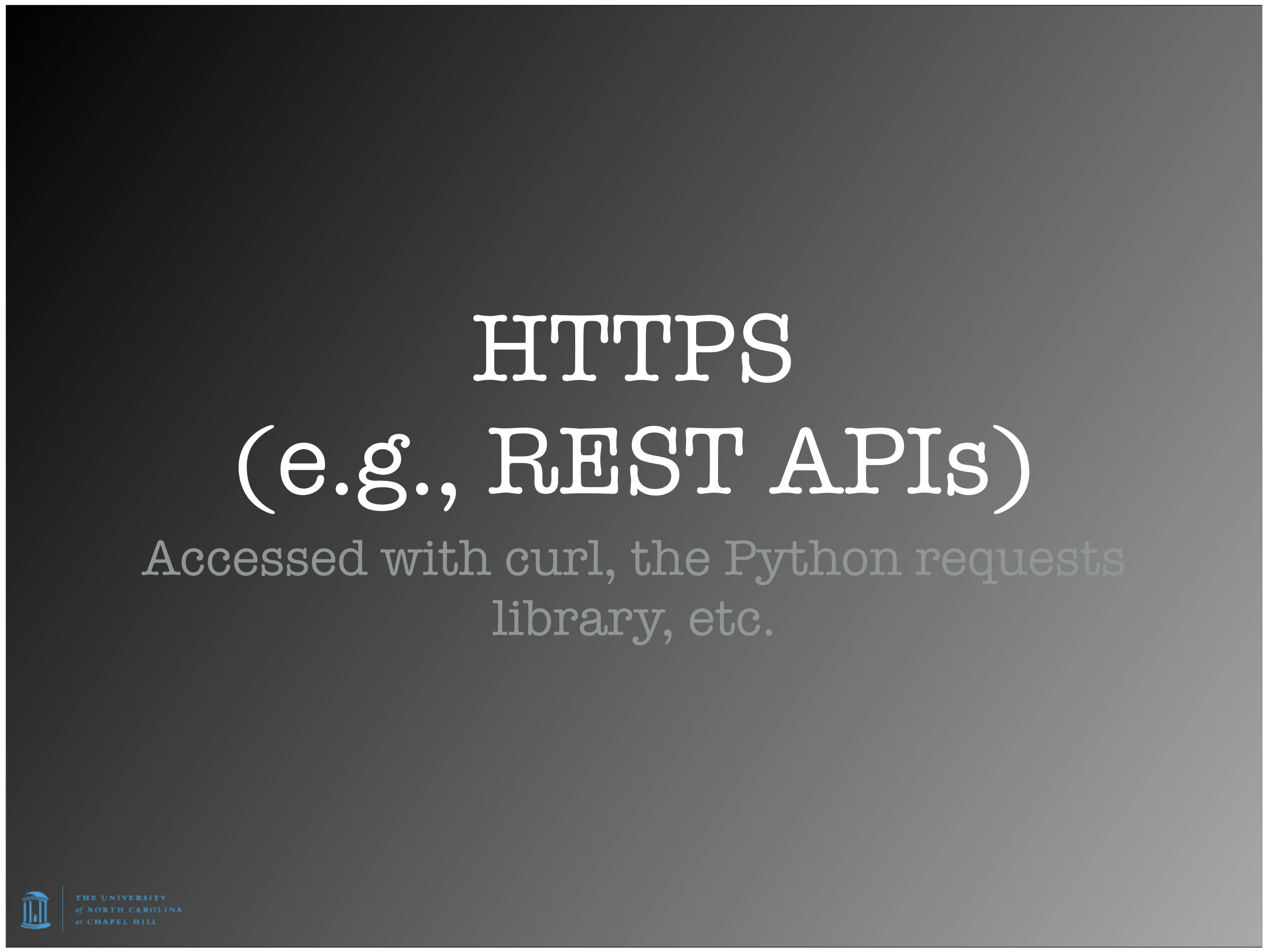
When working in network automation, you often find yourself dealing with PKI whether you realize it or not.



SSH (e.g., RSA keys)

Used by Ansible, Python/Netmiko/etc.

SSH involves the use of RSA keys, for example, which are nothing more than a public/private key pair. If you write Ansible playbooks or use Netmiko, your projects likely involve SSHing to other equipment. If you setup a proper public/private key pair, you can store the public key on all the remote systems so that your host system with the private key can SSH to them without having to use a username/password.



HTTPS (e.g., REST APIs)

Accessed with curl, the Python requests library, etc.

THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

Also, we often make use of REST APIs, whether accessing them on other services or providing them. And REST relies upon HTTPS, which uses SSL/TLS certificates, which again rely on public/private keys.

This is also how browsers work when accessing secure sites. All browsers rely upon a set of stored CA public certificates to determine whether an SSL/TLS certificate was, in fact, digitally signed by a CA.

RPKI

Used to secure the Internet's routing infrastructure, particularly BGP

Thank You

https://frank.seesink.com/presentations/Internet2TechEx-Fall2024/

Frank Seesink
frank@seesink.com
frank@unc.edu

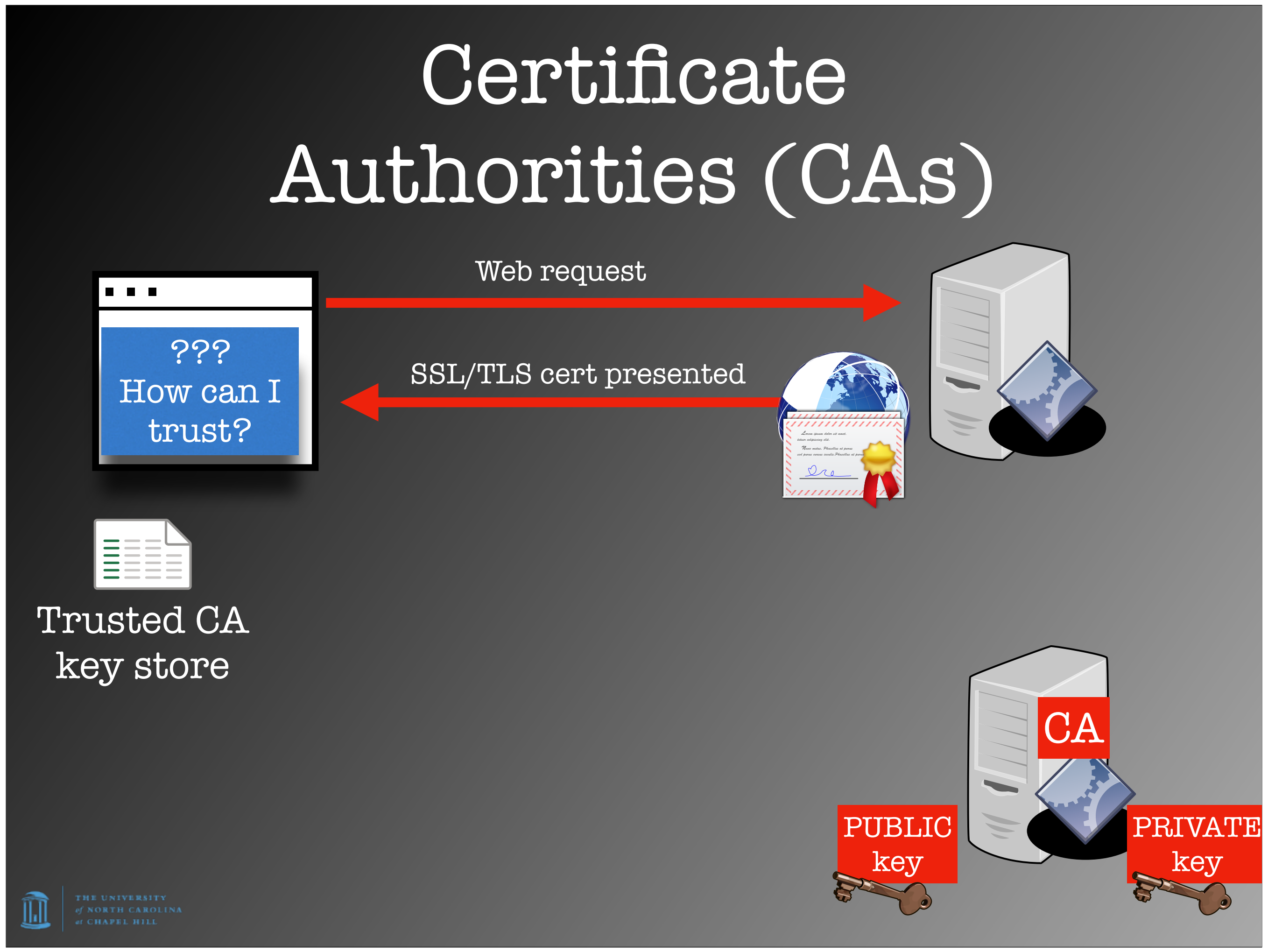
THE UNIVERSITY *of* NORTH CAROLINA *at* CHAPEL HILL

Bonus Material

For those who download the slide deck later, slides I could not fit into the session

THE UNIVERSITY *of* NORTH CAROLINA *at* CHAPEL HILL

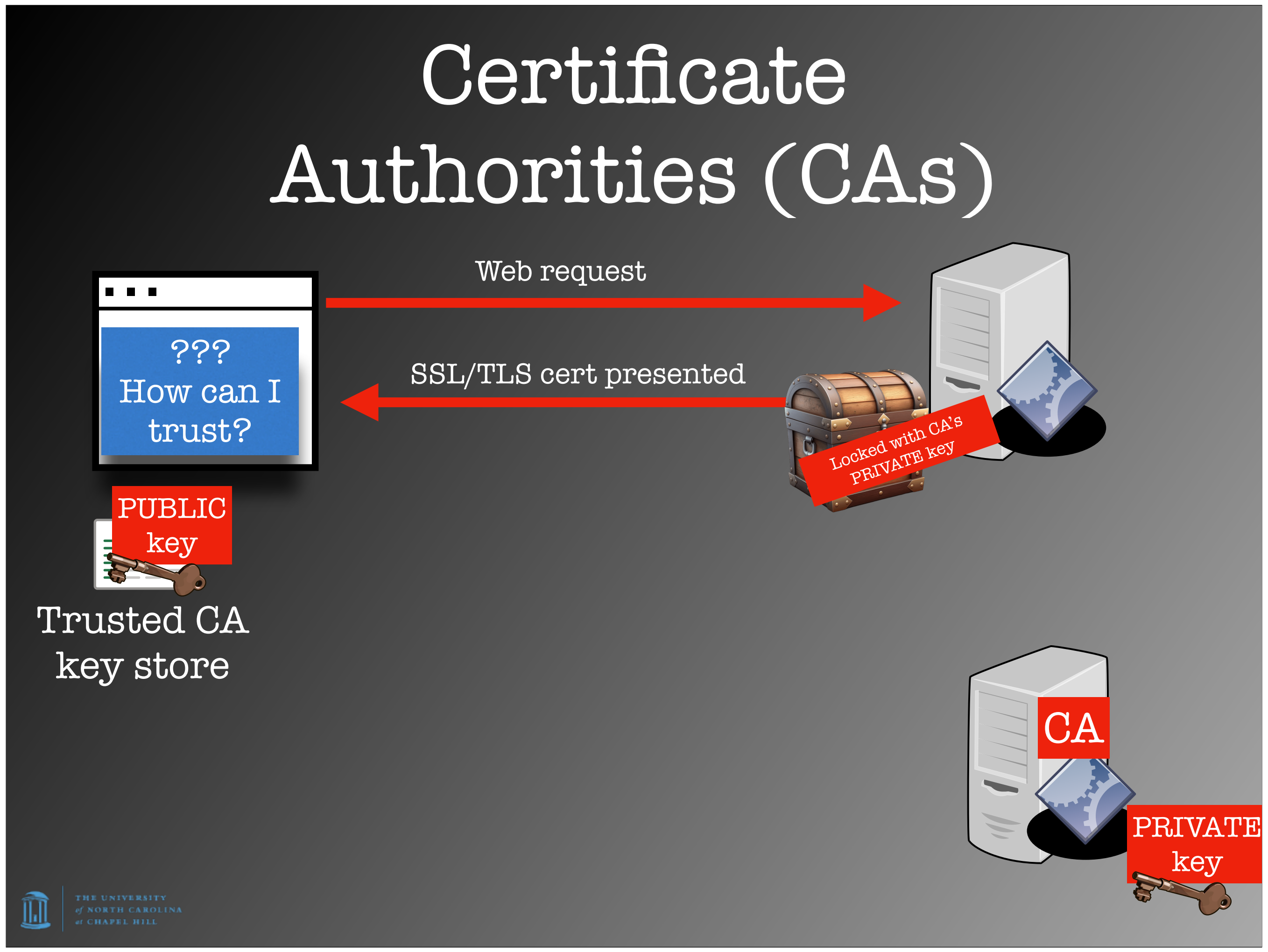
Certificate Authorities (CAs)



When dealing with the Web and REST APIs, we often deal with HTTPS. This involves the use of SSL/TLS certificates. But how exactly do those work? We typically think of the interaction as "browser/client sends request, server sends back certificate". But how can we trust this certificate? Maybe they PhotoShopped it.

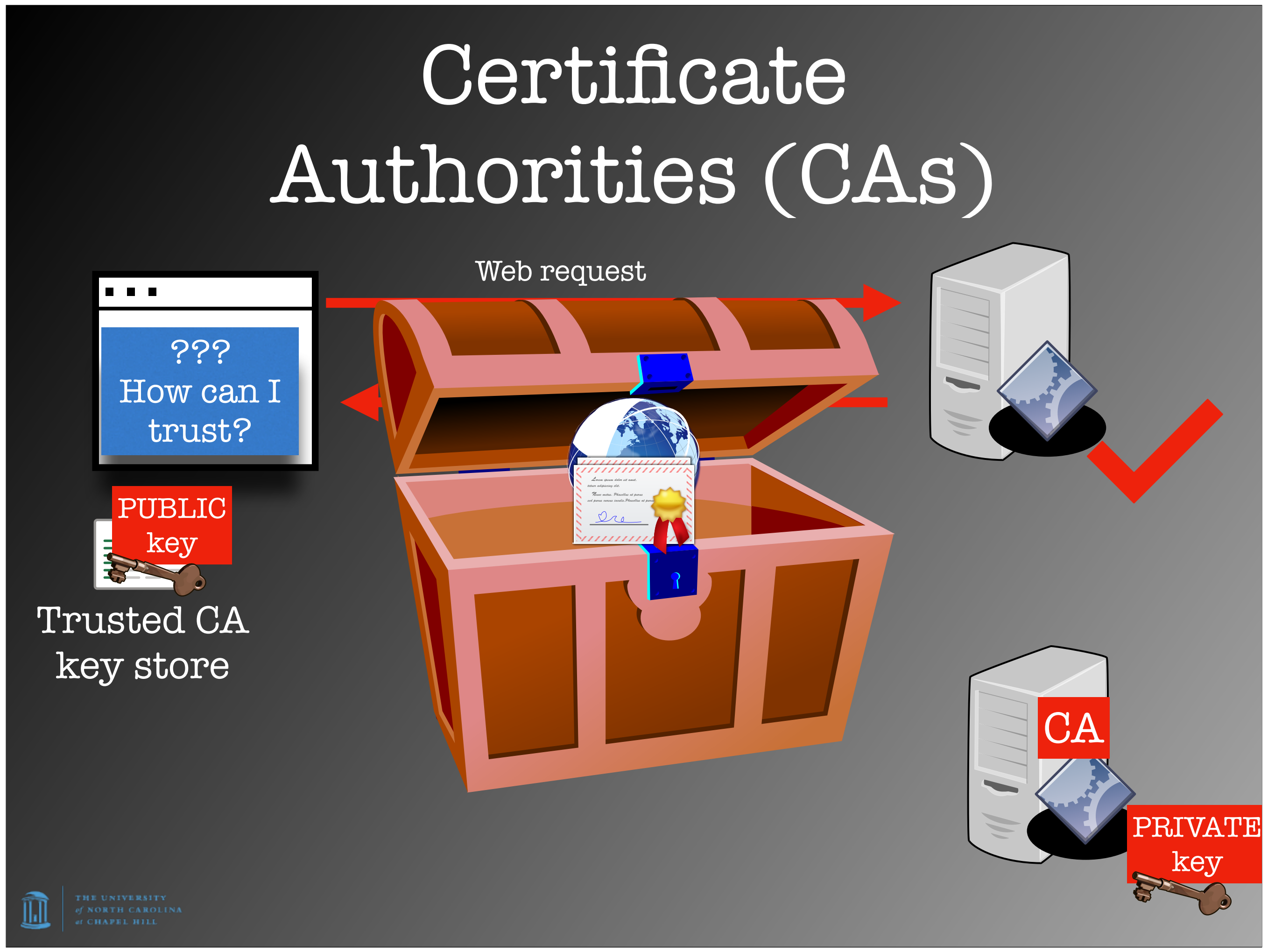
This is where Certificate Authorities (CAs) come in. As their name implies, they are authorities on certificates. In the security world, everything comes down to trust. Who do you trust? If I trust Bob, and Bob says Alice is ok, then I trust Alice. This applies as much in the computer world as anywhere. So how do we establish this "trust" with CAs like Verisign, Comodo, etc.?

Think of it this way. A CA is nothing more than another entity with a private and public key pair. Only in this case, your browser contains a list of trusted CA public keys (i.e., public keys that the browser creators somehow received from the CAs "out of band").



Certificate Authorities (CAs)

Using our treasure chest analogy, now think of an SSL/TLS certificate as nothing more than a document that was “digitally signed” by a CA using its private key. When your browser visits a site, the site presents its SSL/TLS certificate, which says that it was signed by a particular CA. The browser then looks up that CA’s public key within its keystore, which it then uses to unlock the certificate.

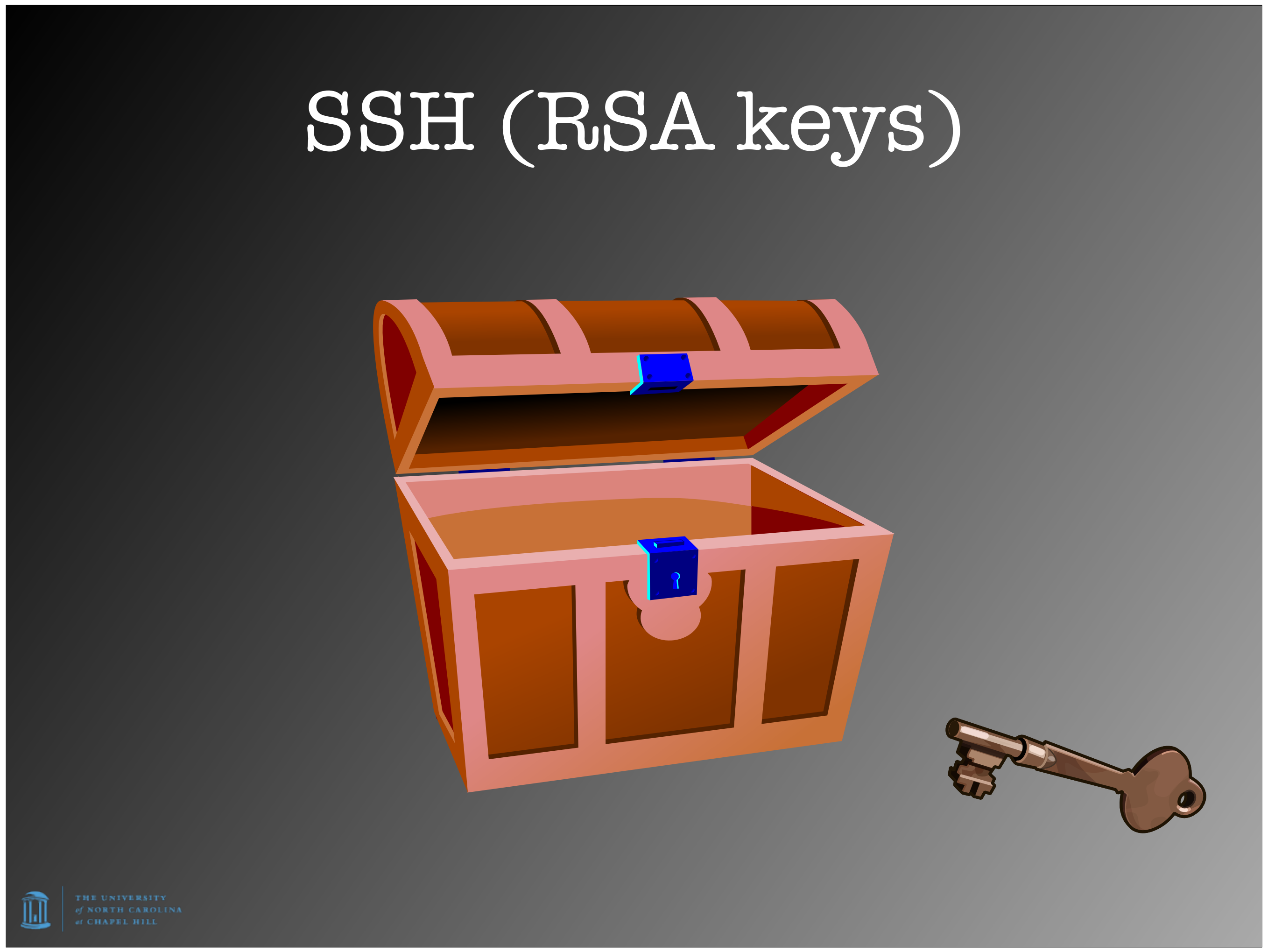


Certificate Authorities (CAs)

If the CA's public key works, the browser then knows that this certificate was signed by that CA, and in turn it can now trust this site. There is more to this as the browser and website then negotiate the encryption to use, but it gets the point across.

SSH

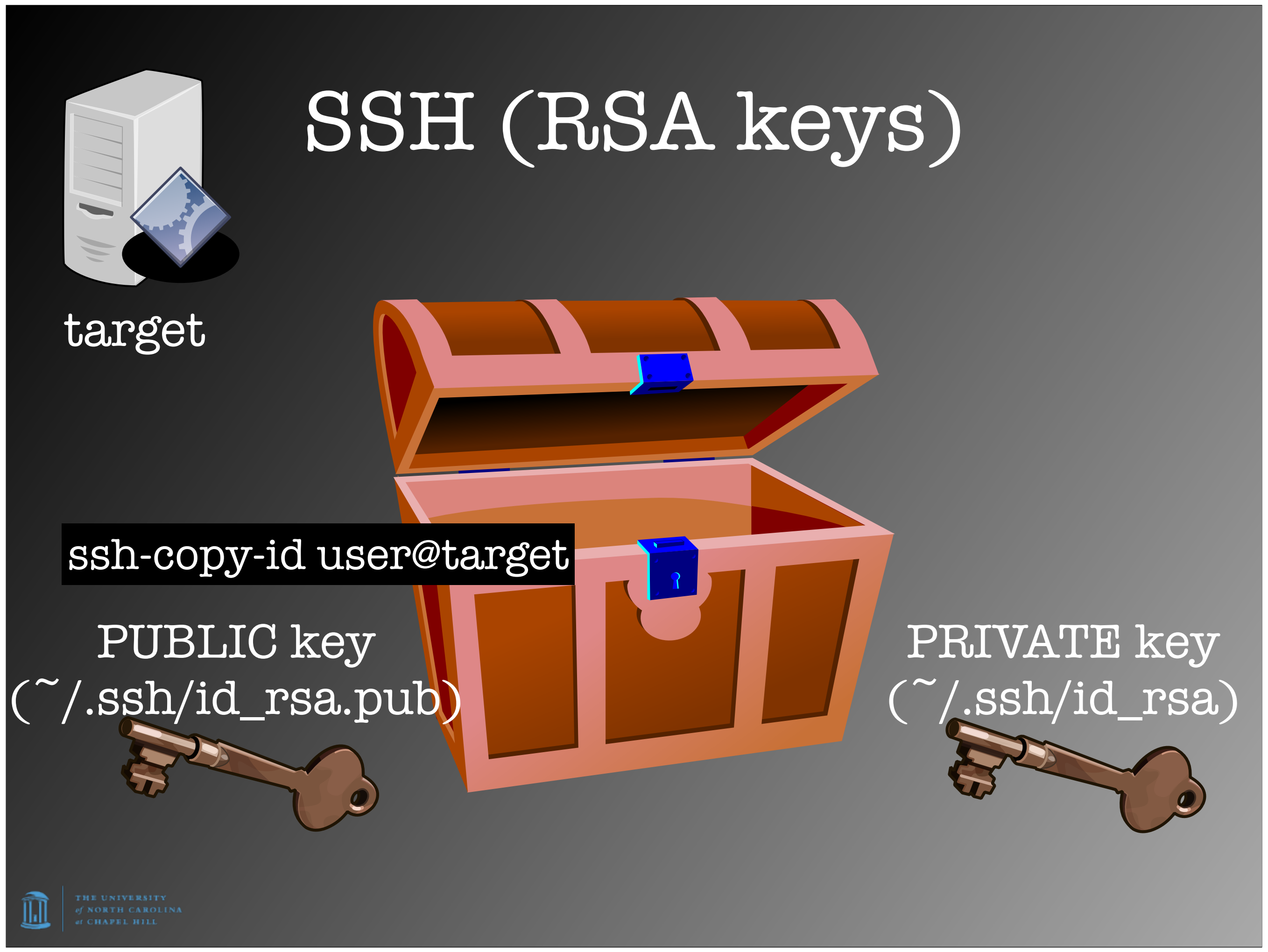
THE UNIVERSITY
of NORTH CAROLINA
at CHAPEL HILL



SSH (RSA keys)

When you work with SSH, the first thing that you need to do is generate a public/private RSA key pair. Typically both the PRIVATE and PUBLIC keys are stored in your home directory in ~/.ssh (e.g., ~/.ssh/id_rsa for the private key, ~/.ssh/id_rsa.pub for the public key).

It is important to note that the PRIVATE key here works just like our analogy. That is, you should not be sharing this with anyone. This is how you prove that “you are you.”

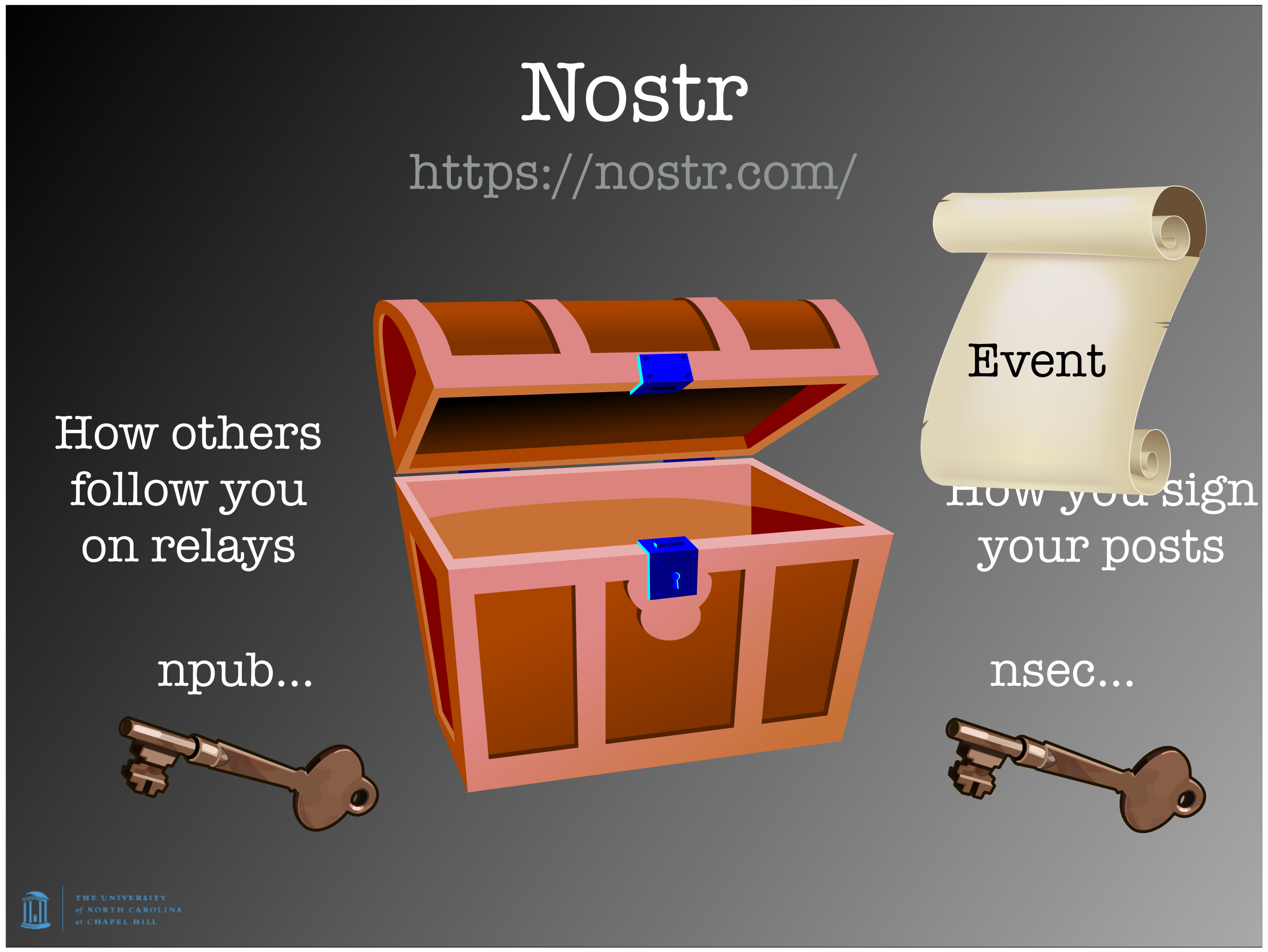


SSH (RSA keys)

The PUBLIC key is just that: public. So you put that out in the world where anyone can get to it. Typically you use a command like ssh-copy-id to SSH into a remote device (target), where you first authenticate with your username/password. This command then copies your PUBLIC key to that target (e.g., if *nix system, typically it adds your PUBLIC key to ~/.ssh/authorized_keys). When later you try to use things like SSH or SCP to go to username@target, you no longer need to provide a password, as the remote side will authenticate you based on a challenge. This challenge involves the remote side putting a random string into a “chest” and locking it with your public key, then asking you to tell it what is inside the chest. Your system uses your private key to unlock it and get that random string back, proving that you, in fact, have the matching private key.

Nostr

THE UNIVERSITY *of* NORTH CAROLINA *at* CHAPEL HILL



Nostr

https://nostr.com/

How others follow you on relays
npub...
Event
How you sign your posts
nsec...
THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

Though not directly related to network automation, other technologies such as Nostr, an alternative to Twitter, also rely on PKI, where you create a public/private key pair, and you digitally sign all of your posts (called “events”) using your private key. The public key is generally presented as a string with the prefix ‘npub’ and the private key with the prefix ‘nsec’. “Following” someone on Nostr is really just tracking them on relays via their public key.