# Digital Signature

Xinwen Fu, Ph.D

Professor

Department of Computer Science

University of Massachusetts Lowell

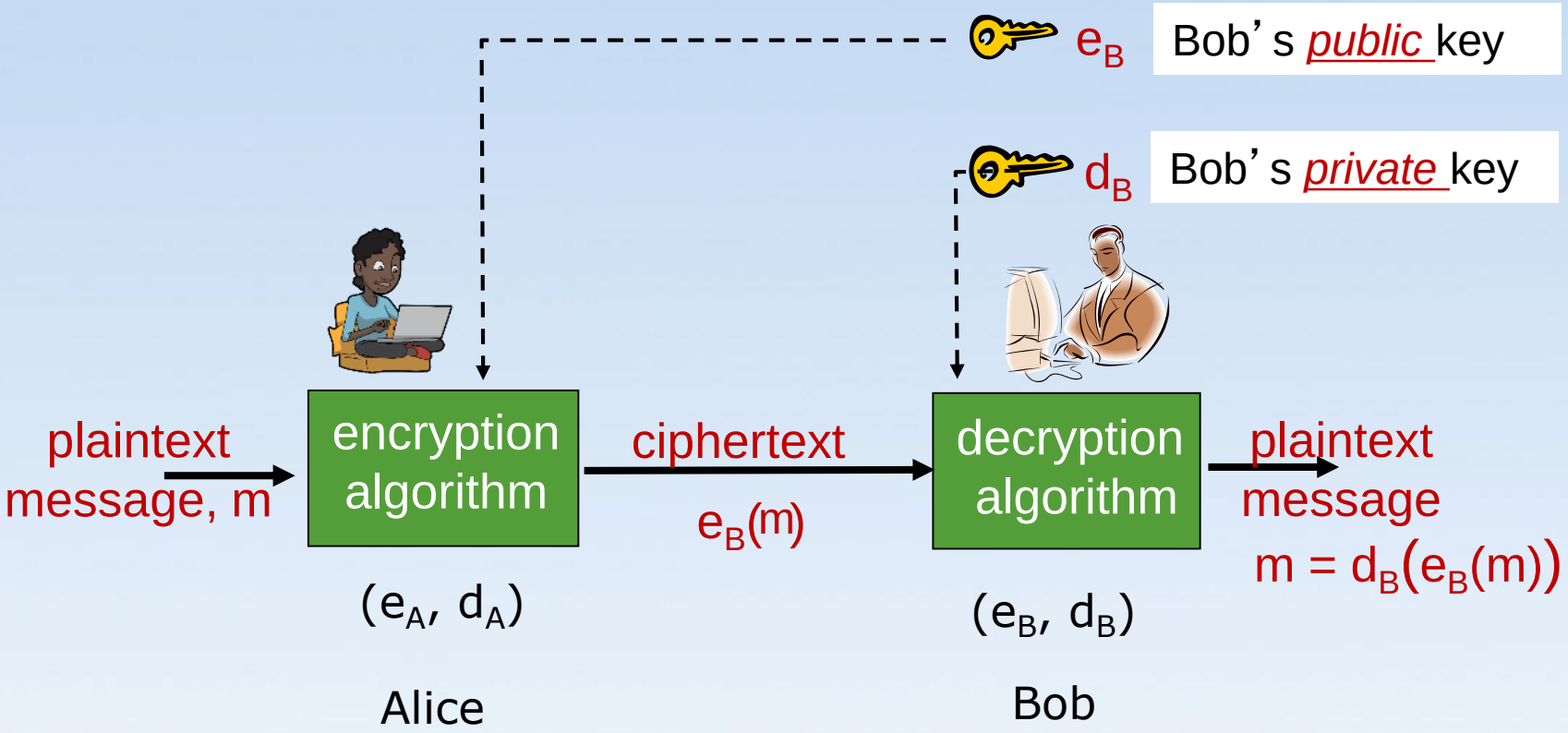
# Outline

- Introduction to digital signature
- Introduction to certificate
- Hands-on labs

# Conventional Signature

- Verifies that the signer approves the writing
- How do we sign a digital document like a pdf file?

# Review of Public Key Cryptography



# Generate Public and Private Key Pair

- Generate public and private key pair
  - openssl *genpkey -out privkey.pem -algorithm rsa*
  - Note: privkey.pem contains both public and private keys
- Extract public key from privkey.pem and save it in pubkey.pem
  - openssl *rsa -in privkey.pem -outform PEM -pubout -out pubkey.pem*

# Encryption with RSA

- Create a file
  - echo "Welcome to LinuxCareer.com" > encrypt.txt
- Encryption
  - openssl *rsautl -encrypt -inkey pubkey.pem -pubin -in encrypt.txt -out encrypt.dat*
- Encode the binary ciphertext with base64
  - openssl *enc -base64 -in encrypt.dat -out encrypt.dat.base64*
  - Note: encoding is not necessary, but needed for sending the ciphertext through our chat server

# Decryption with RSA

- Decode *encrypt.dat.base64* and get the binary ciphertext
  - openssl *enc -base64 -d -in encrypt.dat.base64 -out encrypt.dat*
- Decryption
  - *openssl rsautl -decrypt -inkey privkey.pem -in encrypt.dat -out new_encrypt.txt*

# Public Key Encryption Properties

1. need e(.) and d(.) such that

$$d_B(e_B(m)) = m$$

2. given public key $e_B$, it should be impossible to compute private key $d_B$

*RSA:* Rivest, Shamir, Adelson algorithm

# Another important property

The following property will be *very* useful

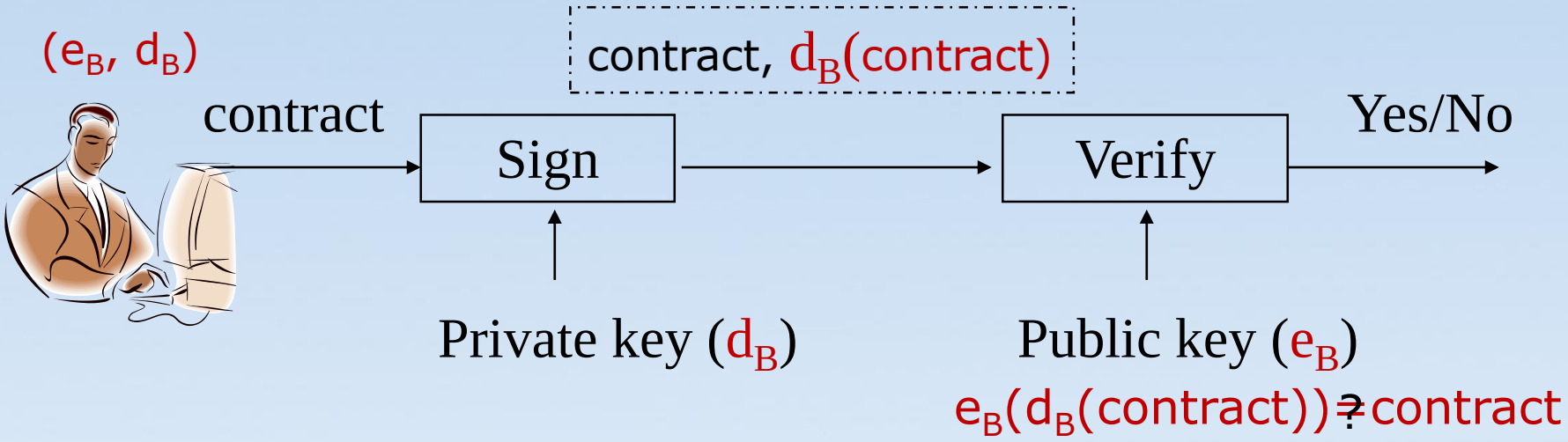
$$e_B(d_B(m)) = m = d_B(e_B(m))$$

use public key first, followed by private key

use private key first, followed by public key

*result is the same!*

# Naive Digital Signature




- Bob encrypts the contract with his private key
  - $d_B$(contract) is the signature
- Those who have Bob's public key can verify the signature
  - Decrypt the signature
  - Compare the decrypted contract with the contract sent over

# Digital Signature Properties



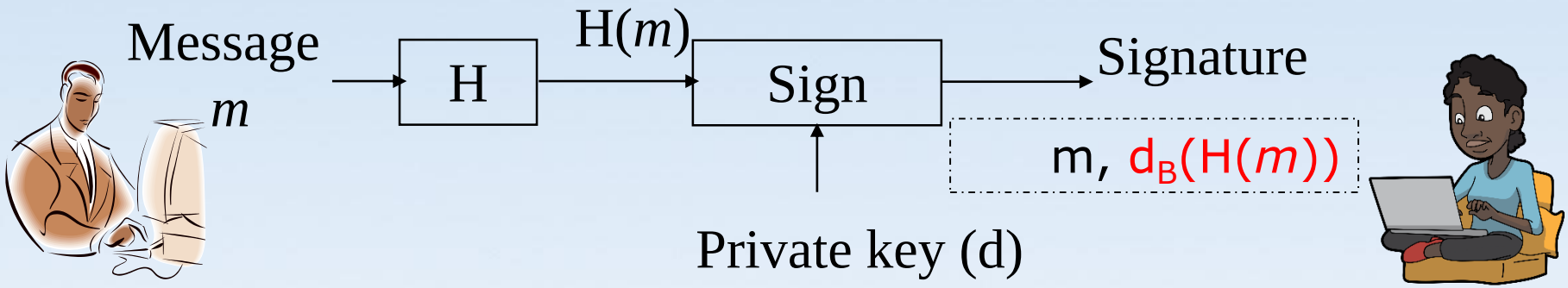

- The party with private key can create digital signature
- Anyone who knows the public key can verify digital signature
- The signer cannot deny that he/she has done so

# Problem of Naive Digital Signature?

- contract, $d_B$(contract)

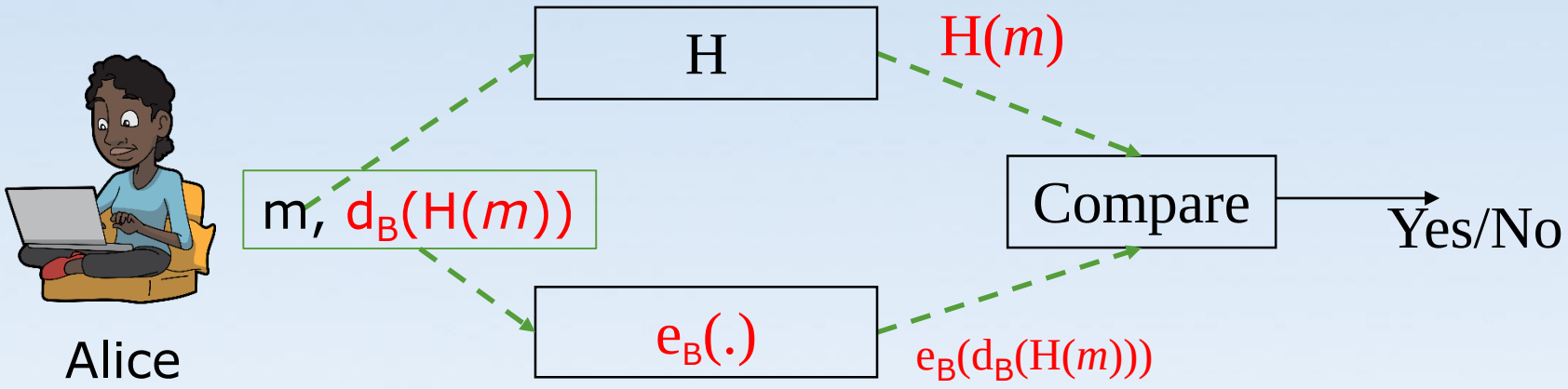
# Signing of true Digital Signature

- Generate message hash
- Encrypt the hash with the private key



# Verification of True Digital Signature

1. Hash the message *m* that was sent over to get H(m)
2. Decrypt digital signature $d_B(H(m))$ with Bob's public key
3. Compare H(m) and $e_B(d_B(H(m)))$

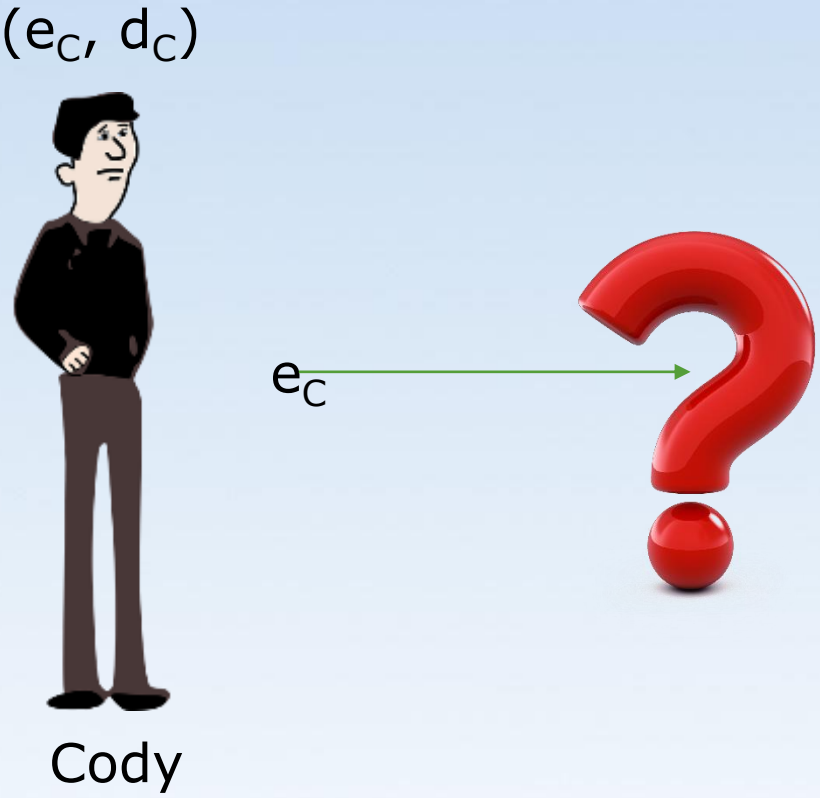


# Outline

- Introduction to digital signature
- Introduction to certificate
- Hands-on labs

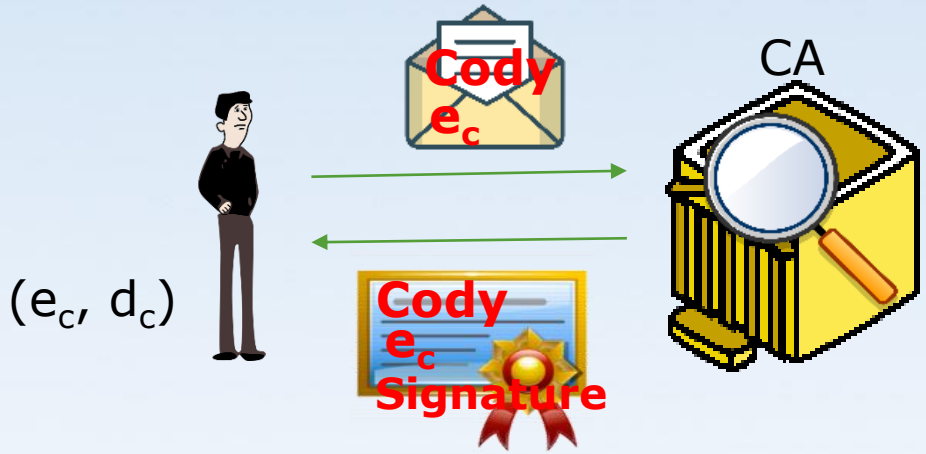
# Question

- How can you safely give your public key to everyone?



# Obtain a Certificate Issued by a certificate authority (CA)

1. Cody generates his key pair ($e_c$, $d_c$)
2. He goes to a CA face-to-face
3. CA verifies his information: email, ID, address, etc.
4. CA issues Cody a certificate, which contains
   - Cody's identity, other information and his public key $e_C$
   - Signed by CA



# Verify a Certificate

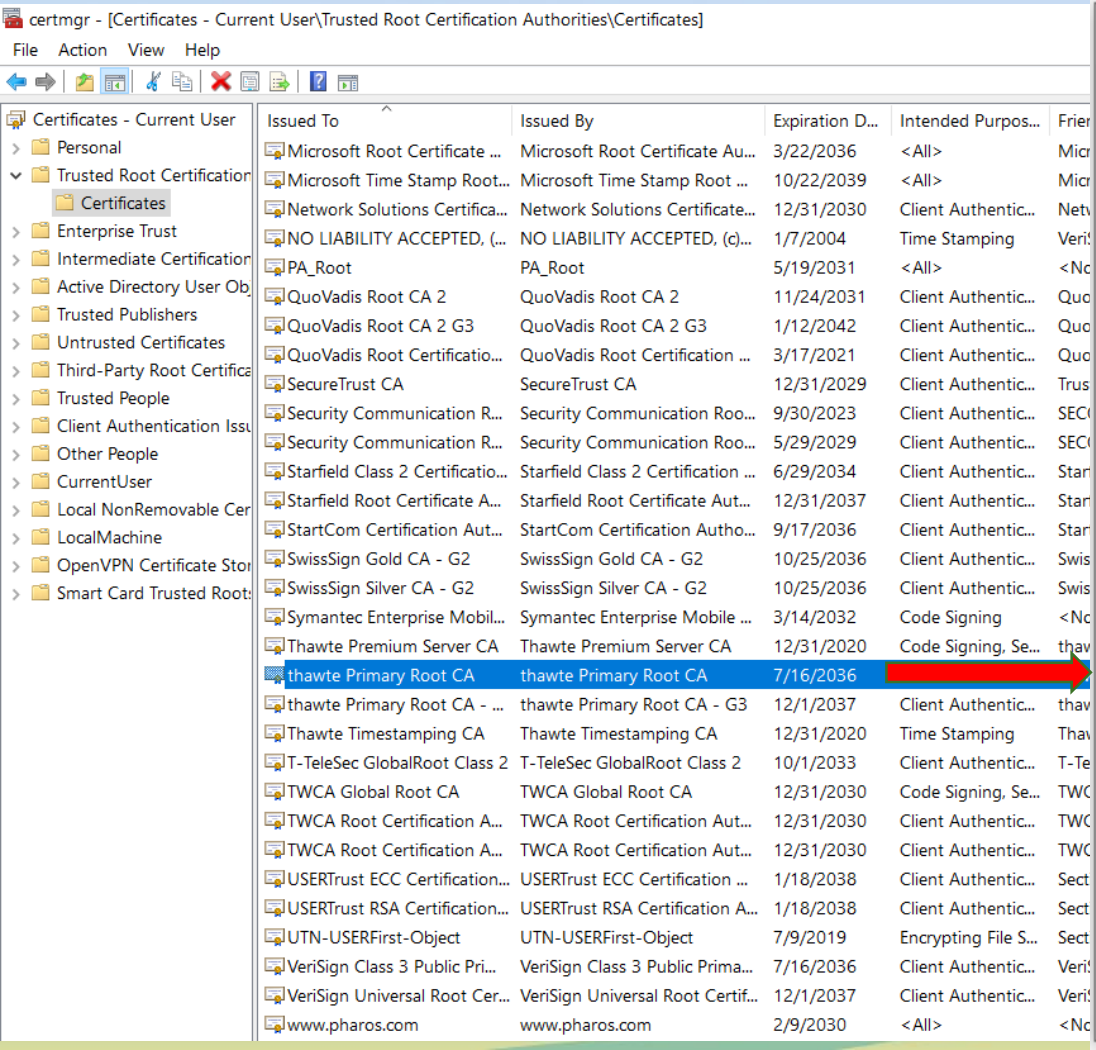
- Cody can get a certificate from a certificate authority (CA)
- Every computer is shipped with the CA's public key in the format of a certificate
- When Cody sends its certificate to a computer, how can the computer verify Cody's public key is genuine?

# Terms for Certificates

- The owner of a certificate is called a subject
- Common name is the identity of the owner
- Subject's *CommonName* can be:
  - An explicit name, e.g. cs.uml, or
  - A name with a wildcard character, e.g. *.uml.edu

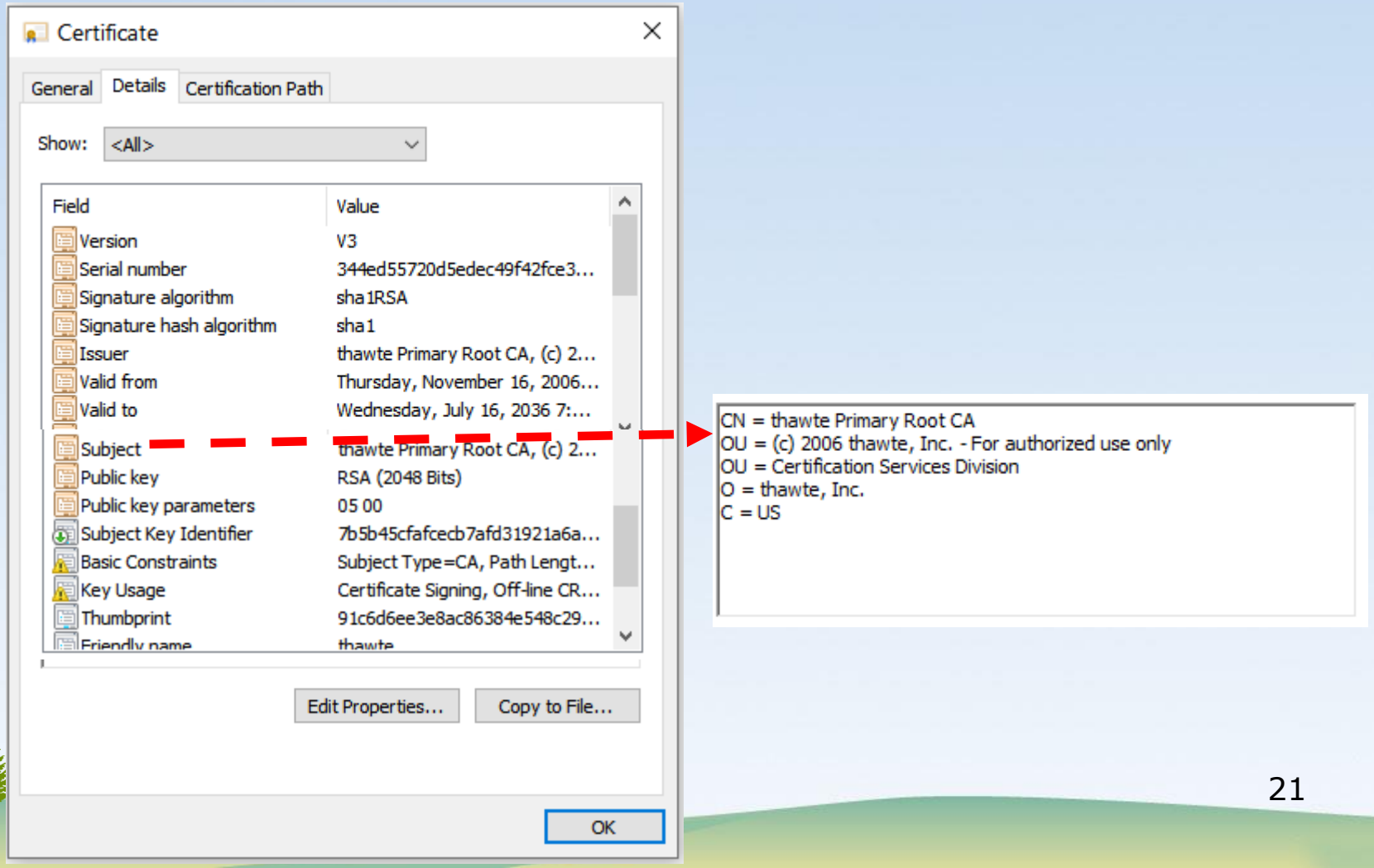
# Windows 10 Certificate Store

- *Type here to search -> certmgr.msc*



# Example Certificate Details

- Click subject



# Kali Certificate Store

- /etc/ssl/certs
- Command to view a certificate
  - openssl x509 -in certificate-name.pem -text

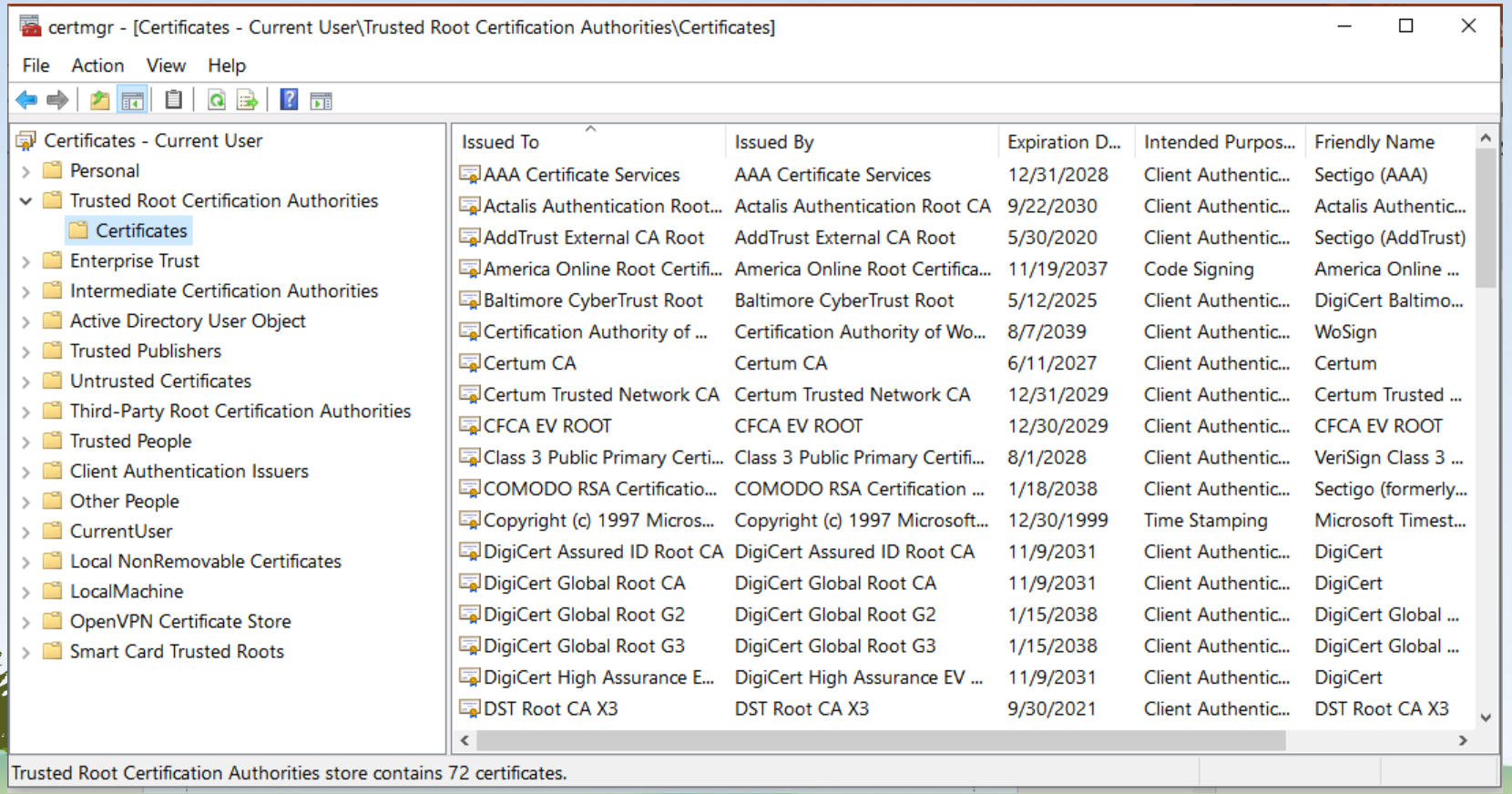
# Outline

- Introduction to digital signature
- Introduction to certificate
- Hands-on labs
  - Windows 10 certificate store
  - Sign and verify a file locally
  - Sign and verify a message through chat server

# Certificate Store

# Windows 10 Certificate Store

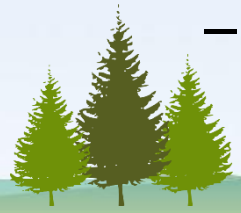
- TA finds the Windows 10 certificate store
- All people check content of a certificate from Trusted Root Certificate Authorities
- Group discussion: why are these certificates needed?



# Digital Signature

# Generate Public and Private Key Pair

- If done before, no need
- Generate public and private key pair
  - *openssl genpkey -out privkey.pem -algorithm rsa*
  - Note: privkey.pem can be used as the private key although it contains the public key
- Extract the public key from privkey.pem
  - *openssl rsa -in privkey.pem -outform PEM -pubout -out pubkey.pem*
- Publish your *pubkey.pem*, e.g. to our chat server
  - Never share *privkey.pem*



# Sign a file at Your Computer

- Sign a file called changelog or messages saved in a file
  - *openssl dgst -sha256 -sign privkey.pem -out sign.sha256 changelog*
  - The output sign.sha256 is binary
- Encode the binary signature with base64
  - *openssl enc -base64 -in sign.sha256 -out sign.sha256.base64*
  - Not really needed. It is needed here since we can send the base64 encoded message to our chat server
- Send both message and base64 encoded signature to our chat server

# Verify the Signature

- Save received signature into a file, e.g., called sign.sha256.base64
- Decode sign.sha256.base64 and get the binary signature
  - *openssl enc -base64 -d -in sign.sha256.base64 -out sign.sha256*
- Verify the signature with the public key
  - *openssl dgst -sha256 -verify pubkey.pem -signature sign.sha256 changelog*

# Message Signature via Chat Server

- Sender generates public and private key pair (no need to do it if already done)
- Sender sends the public key (content of pubkey.pem) to Receiver
  - Receiver saves the public key into a file, e.g. called pubkey.pem
- Sender sends a message to receiver and saves the message into a file called, e.g., msg
  - Receiver saves the received message into a file called, e.g., msg
- Sender signs its msg, generate the signature file and encodes it with base64, e.g. called *sign.sha256.base64*
- Sender sends the content of *sign.sha256.base64* to Receiver
  - Receiver saves it into a file, e.g., called *sign.sha256.base64*
- Receiver decodes *sign.sha256.base64* into the binary signature file and verifies it

# Message under MITM

- Assume only the message is changed by MITM
  - That is, hahaha… is added to the received message
- Repeat the same procedure
- Is the verification ok this time?