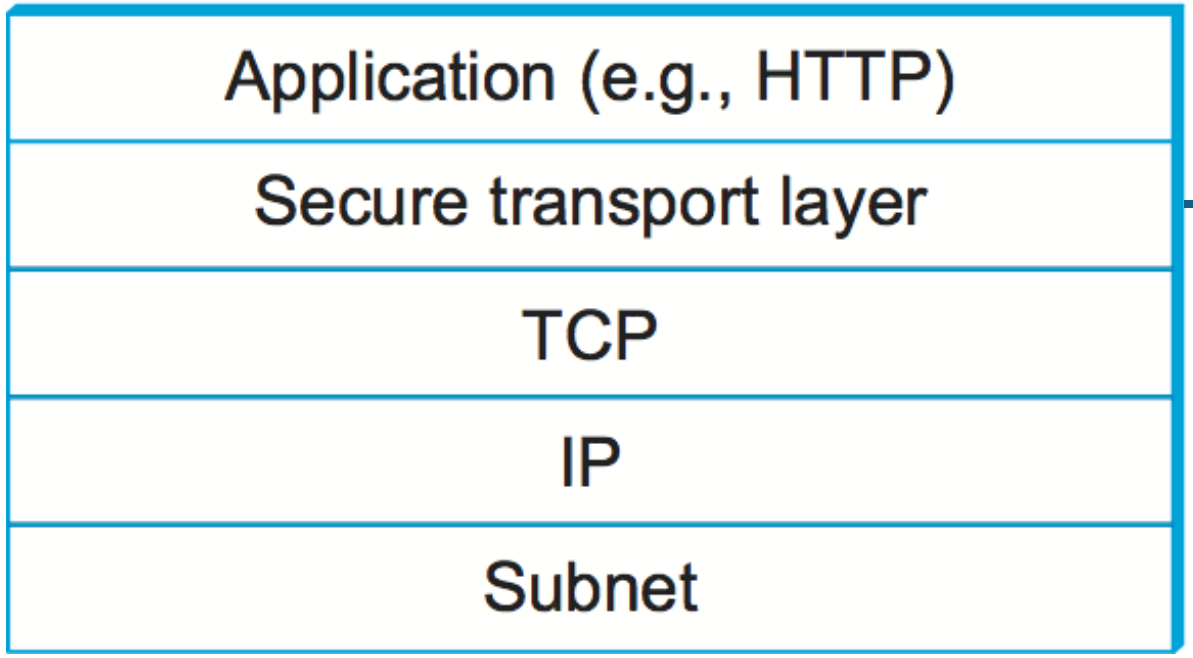
Lecture 22 – Practical aspects of security

Lecturer: Venkat Arun

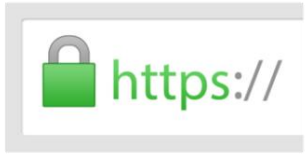
Securing a web connection



Also known as TLS or SSL

If this is used with HTTP, we call it HTTPS.

This is what this icon on your browser means:



Two steps in establishing a secure connection

1. The client's browser needs to know which public key they can trust for the website's domain name
   - E.g. if I am visiting bankofamerica.com, I need to be sure that the public key based on which I am authenticating the website belongs to the actual bank. Otherwise, I will give my login password to someone else!
   - While technically straightforward, maintaining trust is extremely hard
2. Establish a secure connection given this public key
   - We'll study step 2 first

Step 2: Establishing a secure connection

Assuming there is a way for the client to trust the public key

Step 2: Establish a secure connection

Here is the list of cryptosystems I support: RSA, Elliptic curve version X, post quantum secure systems…

Client

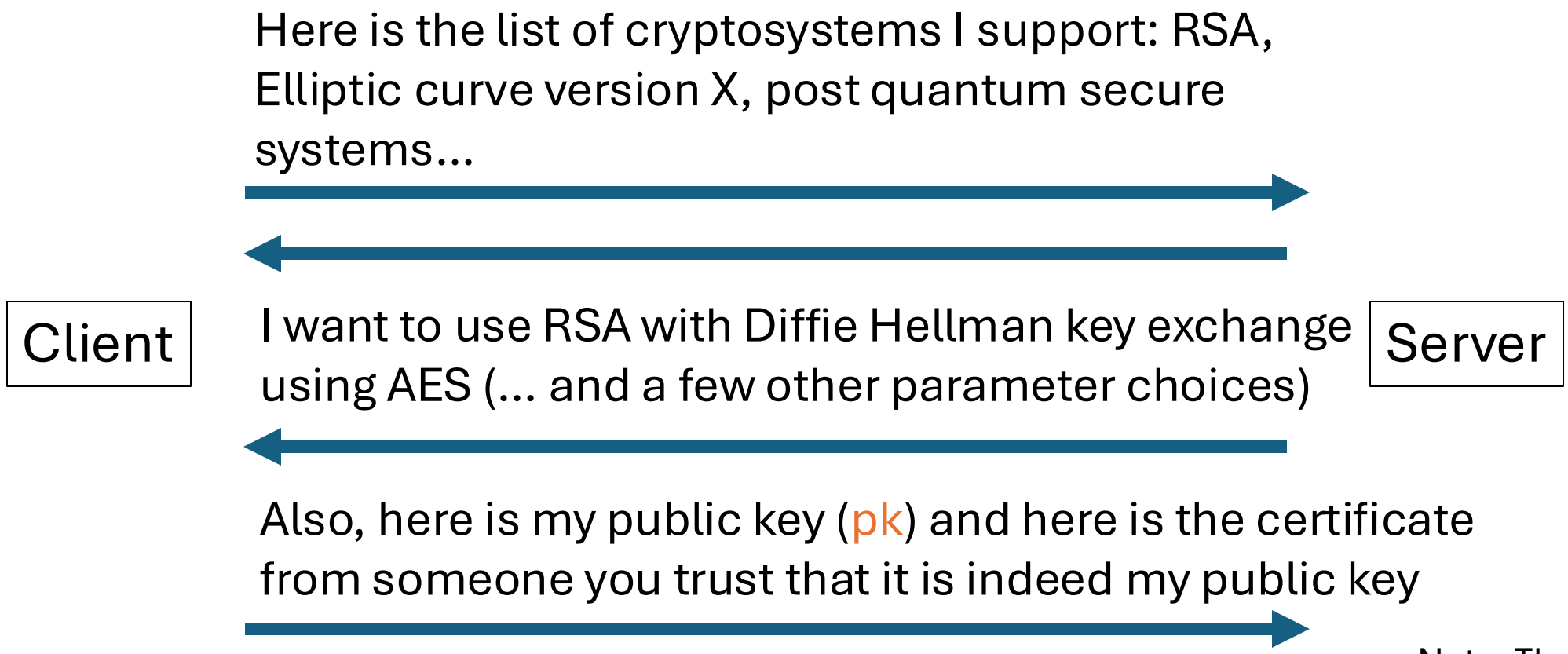
I want to use RSA with Diffie Hellman key exchange using AES (… and a few other parameter choices)

Server

It is important that both parties use a cryptosystem they are comfortable with. This allows for the internet to slowly (or quickly) move to new systems when old ones become compromised

For example, the MD5 hash function was broken in 2005. Similarly, quantum computers will break RSA if and when they become real

Step 2: Establish a secure connection

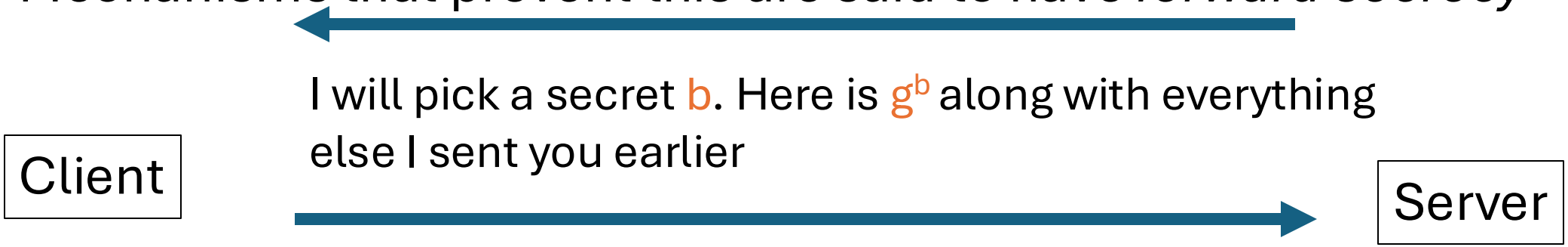


Note: The "SSH authentication" shown in the previous class was incomplete and susceptible to the MitM attack

Problem: Forward secrecy

If someone stores a transcript of the previous communication, and later the server's private key (sk) becomes compromised, they can decrypt everything

Mechanisms that prevent this are said to have *forward secrecy*



Let's do a Diffie Hellman key exchange. I'll pick a secret a. I'll give you $Enc_{pk}(g^a)$

Now, g^{ab} can become our secret key which we will use for symmetric encryption. Even if someone later learns sk, they can learn g^a and g^b but not g^{ab}

Problem: RSA is slow

- The prime numbers involved are often 2048 bits long. It takes ~1 ms to exponentiate such a large number on a modern computer

- This is why we used the DH key exchange to ensure forward secrecy. DH can often work with 160-bit keys and is hence faster
  - This is only true when using elliptic curves, which we have not discussed in this class. The method we discussed 2 lectures ago is just as slow as RSA

- However, DH is also slow. This is why we transition to a symmetric encryption scheme like AES for encrypting the actual data. The other schemes are used only for connection establishment

Step 2: Establishing trust in the public key

Suppose the client is talking to a server that claims to be X (say, google.com)

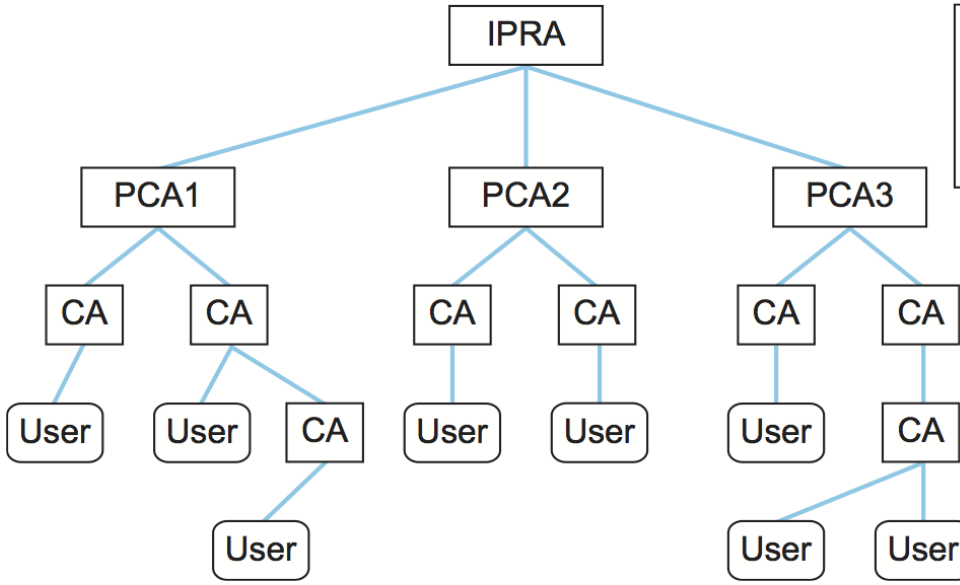
The server authenticates itself using pk.

How does the client know that pk belong to X?

Step 1: How does the browser know what pk to trust?

- Internet policy Registration Authority (IPRA): Sets the standards for everyone to follow. Doesn't actually sign anything
- Policy/Root Certification Authority (PCA/RCA): Web browsers are pre-installed with a set of PCA's public keys that can be trusted. The corresponding private key must be kept secure for a decade or more. Thus, it is kept offline most of the time and brought online only when a new CA's public key needs to be signed.
- CA: Signs the public keys of individual websites. Anyone who owns a domain name can pick a new public key and request a CA to sign it after proving that they indeed own the domain name (e.g. using human verification)

This system is called a **Public Key Infrastructure**



Note: There is a difference between an RCA and a PCA, but we will ignore it

How do we digitally sign things?

Recap:

- Suppose we want to sign something with the key pair (pk, sk).
- sk is two prime numbers (p, q) and a public choice of e
- pk is $n = pq$ and d such that $ed = 1 \; mod \; \phi(n)$
- If we have a message m represented as an integer in the range 1 < m < n, we can sign it as $\text{sign} = m^d \; mod \; n$ and send (sign, m) to the verifier
- The verifier knows pk and checks the signature using $(sign)^e \; mod \; n == m$

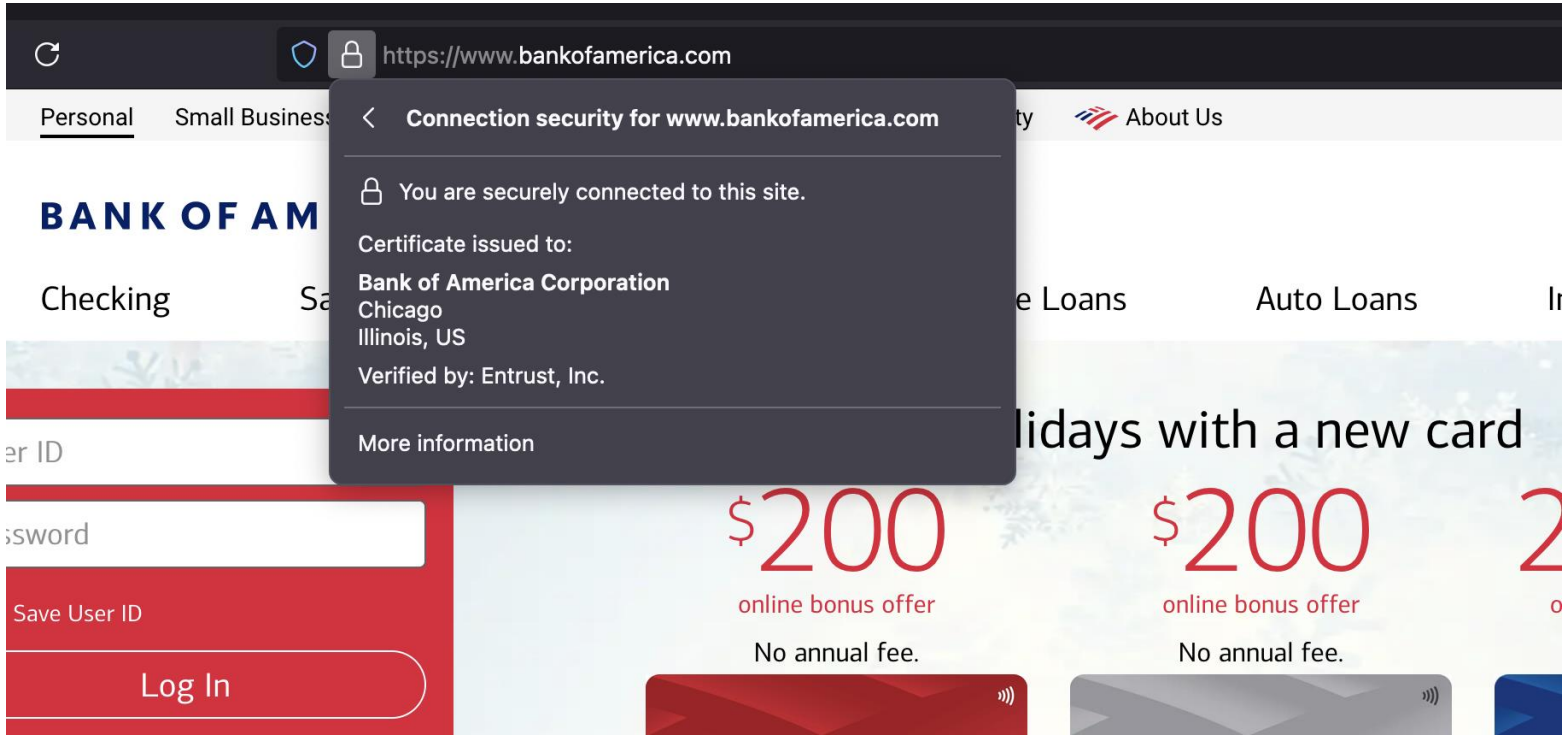
How do we digitally sign things?

- If the message is not an integer in the range (1, n), we can convert it to such an integer using a collision resistant hash function such as SHA256
- A collision resistant hash function, H, is a deterministic function that takes an arbitrary sequence of bytes as input and produces an output in a fixed range (say, a 256-bit number)
  - It guarantees that it is computationally hard to find a *collision*. That is, find $x \neq y$ such that $H(x) = H(y)$ even though such inputs certainly exist (why?)
  - This means that, if someone signed H(x), it is impossible for an adversary to claim they actually signed a different message by finding a collision
- Linux and OSX machines usually have a program called “shasum” that can compute the hash function of any file
  - Live demo

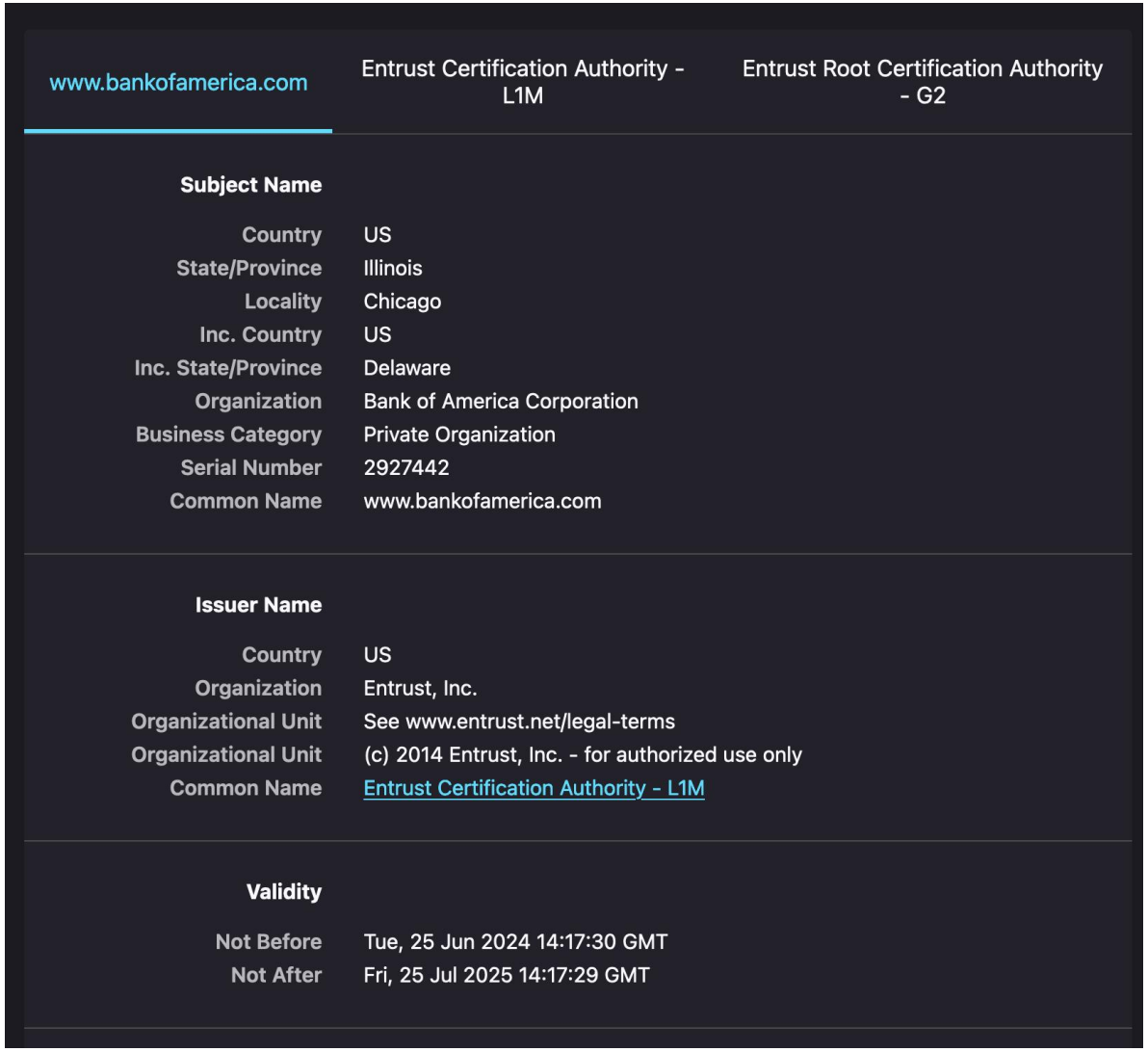
Live Demo: Inspecting a certificate

Entrust is a certificate authority whose public key my browser trusts

Entrust “signs” Bank of America’s public key. It links the key to the domain name “bankofamerica.com” and the legal entity in the US



Live Demo: Contents of a certificate



www.bankofamerica.com | Entrust Certification Authority - L1M | Entrust Root Certification Authority - G2

Subject Name

Country	US
State/Province	Illinois
Locality	Chicago
Inc. Country	US
Inc. State/Province	Delaware
Organization	Bank of America Corporation
Business Category	Private Organization
Serial Number	2927442
Common Name	www.bankofamerica.com

Issuer Name

Country	US
Organization	Entrust, Inc.
Organizational Unit	See www.entrust.net/legal-terms
Organizational Unit	(c) 2014 Entrust, Inc. - for authorized use only
Common Name	Entrust Certification Authority - L1M

Validity

Not Before	Tue, 25 Jun 2024 14:17:30 GMT
Not After	Fri, 25 Jul 2025 14:17:29 GMT

Who guards the guards? How do I know I can trust the Certification Authority? By asking the root Certification Authority of course!

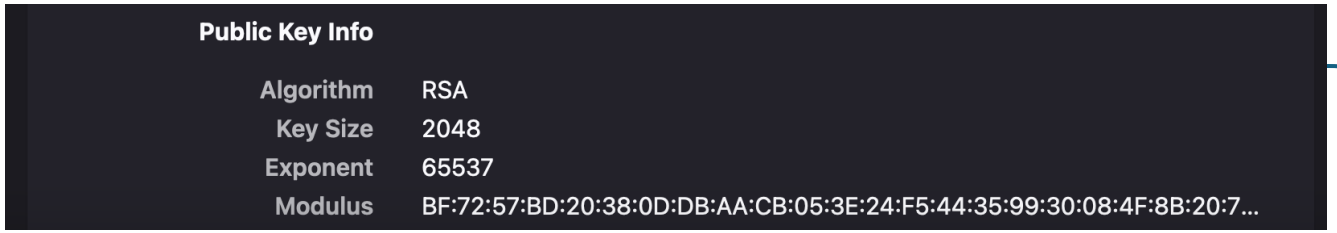
Semi-legal info

Not all CAs do a deep legal verification. For example, LetsEncrypt, another CA just technically verifies whether the entity registering controls the domain name.

This is fine for most websites, but if my bank used it, I'd be scared (personal opinion)

All certificates have a validity period. By keeping this short, we can ensure that even if the secret key is compromised, its effects do not last very long.

Live Demo: Contents of a certificate (…contd)



The technical information about the public key

“Modulus” is the n = pq we discussed in the last lecture

“Exponent” is the e we discussed in the last lecture

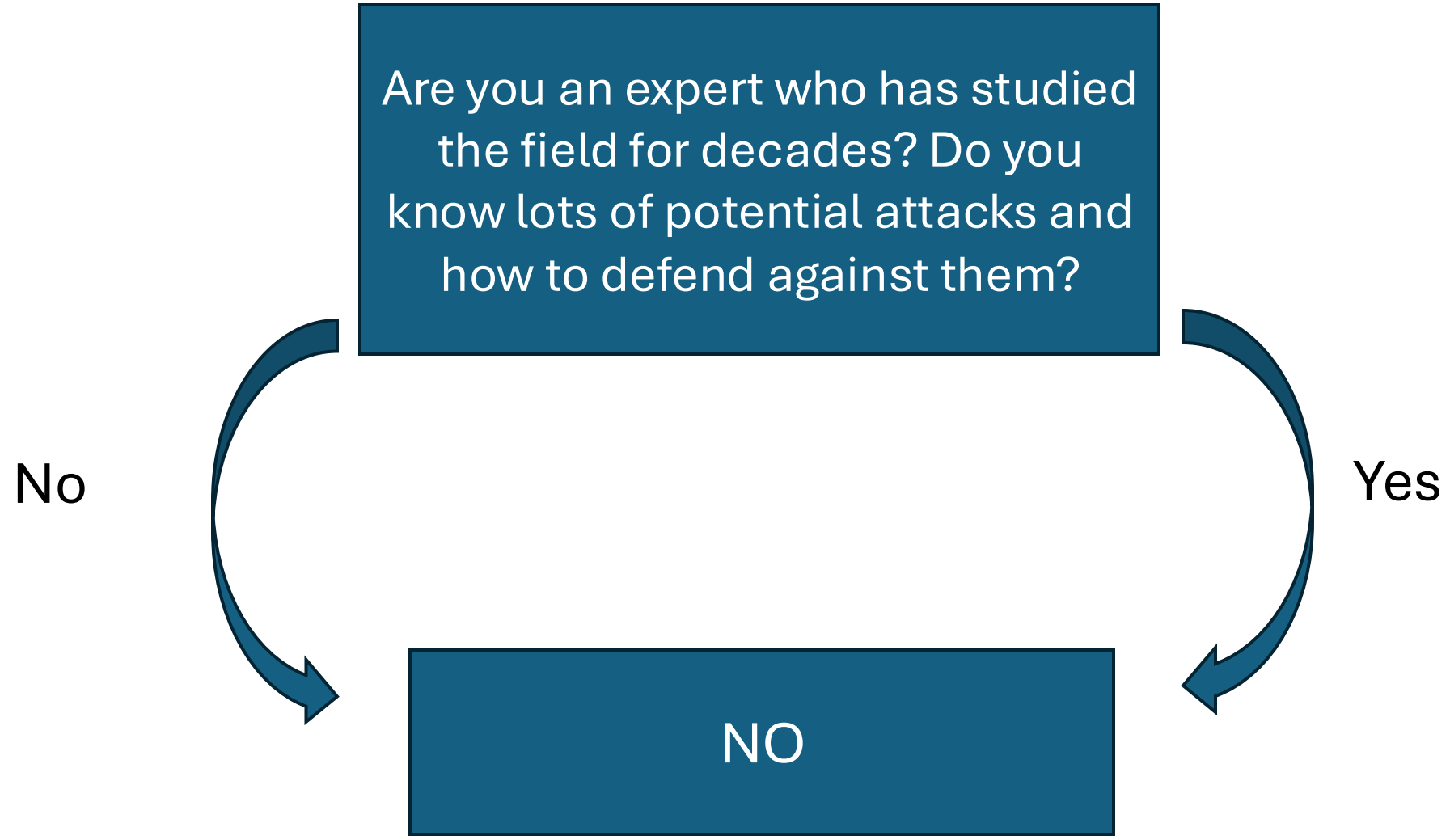
Live Demo: Other notes

- If you check the validity period for the certificate issued to the bank, it is shorter than the one issued to the CA, which is in turn shorter than the one the RCA issued to itself (this self-issued certificate came pre-installed with the web browser)

- Surprisingly, and worryingly, the validity for bankofamerica.com is shorter than google.com!

- If secret keys get compromised, there are ways to **revoke** certificates before the expiration date (called “valid before” in the previous slide). These mechanisms are not perfect

Should you implement custom cryptography?



Assignment 5

- Will ask you to implement custom cryptography :)

Why you should not implement your own custom cryptography?

Note: the following slide will not be a part of any exam.

For the final assignment, you need to be aware of some of the pitfalls, which will be explained in the document

Examples of pitfalls in the RSA cryptosystem

- **Pitfall 1:** If you encrypt a message m by simply computing $e^m \ mod \ n$, people without sk cannot decrypt your message. However, they can tell when you encrypt the same message twice
  - The solution is to add some random bits to m so that you never encrypt the same message
- **Pitfall 2:** If you encrypt the same message n times using enough *different* public keys, it is not possible to detect duplicates. However, if you do this enough times, it is possible to fully recover the key!
- **Pitfall 3:** The exact way in which randomness is added matters and has been used to create attacks
- **Pitfall 4:** The amount of time it takes to encrypt a message can reveal the message or even the secret key. These are called *timing side channel* attacks
- **Pitfall 5:** The power consumed when encrypting/decrypting has also been used to recover secret keys

Security vulnerabilities without breaking any cryptography

This slide may appear in the quiz

- Suppose a website has a comments section. If they are not careful, I can insert code into my comment. When someone else views that comment, my code executes. This is malicious code executing inside a website verified with HTTP. There are many variants of code injection attacks. The standard solution is to “sanitize all inputs” (discussed on the board)
- Buffer overflow attacks can reveal the private key. An example is HeartBleed (discuss on the board)