WILLIAM & MARY
CHARTERED 1693

# CSCI 667: Concepts of Computer Security

Lecture 6

Prof. Adwait Nadkarni

Derived from slides by William Enck, Micah Sherr and Patrick McDaniel

# Reminder: HOMEWORK_2

- Due Oct 3rd
  - Last minute: Bad idea!
- Piazza
  - Ask for clarifications
  - Do not give out answers
  - Collaborate *offline* (meet, email, post, carrier pigeons, …)
  - 25% penalty for late submissions in the first 24 hours
  - 100% penalty 24 hours after deadline.
  - Use LaTeX.
  - In the submission, *maintain symbols*: $R_i$ != $R^i$

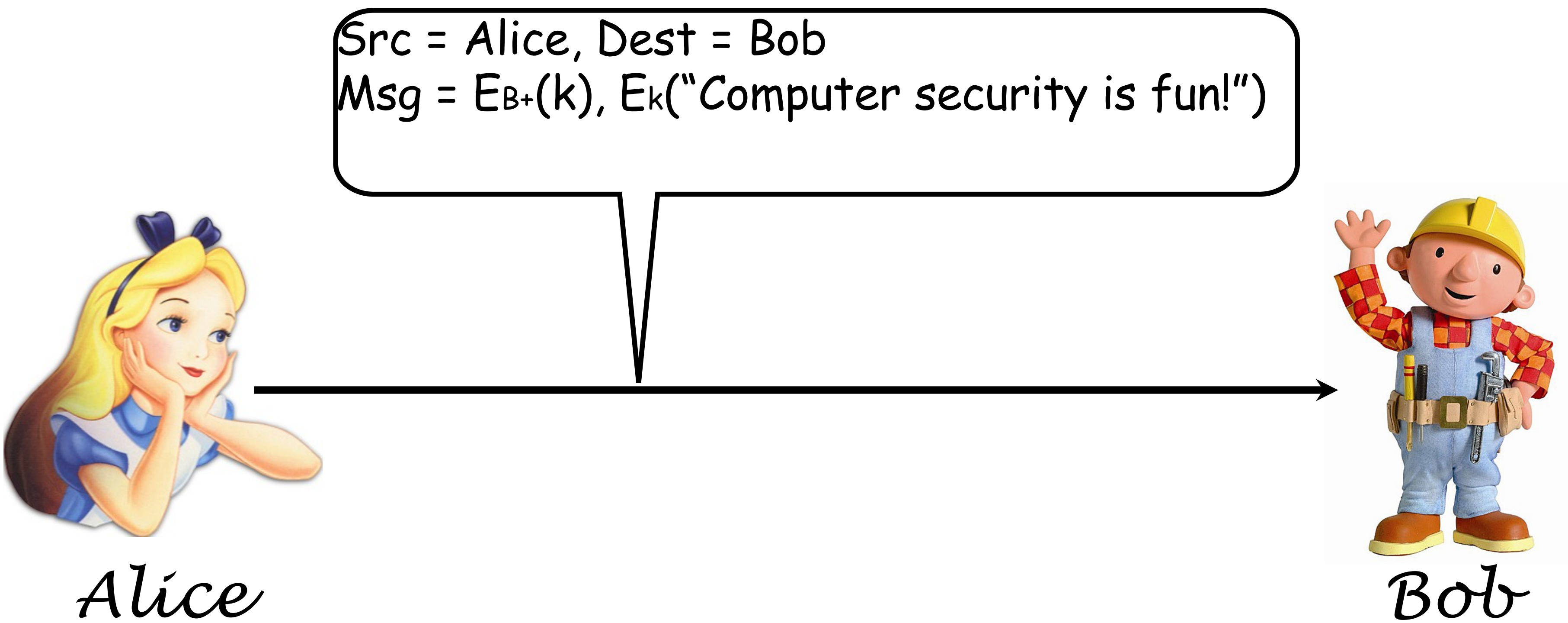
# RSA

- Most public key systems use at least 2048-bit keys
  - Key size not comparable to symmetric key algorithms
- RSA is much slower than most symmetric crypto algorithms
  - AES: ~161 MB/s
  - RSA: ~82 KB/s
  - This is **too** slow to use for modern network communication!
  - Solution: Use hybrid encryption

# Hybrid Cryptosystems

- In practice, public-key cryptography is used to secure and distribute session keys.
- These keys are used with symmetric algorithms for communication.
- Sender generates a random session key, encrypts it using receiver's public key and sends it.
- Receiver decrypts the message to recover the session key.
- Both encrypt/decrypt their communications using the same key.
- Key is destroyed in the end.

# Hybrid Cryptosystems




$(B^+, B^-)$ is Bob's long-term public-private key pair.

k is the session key; sometimes called the **ephemeral key**.

# Public Key Crypto
## (10,000 ft view)

- Separate keys for encryption and decryption
  - Public key: anyone can know this
  - Private key: kept confidential
- Anyone can encrypt a message to you using your public key
- The private key (kept confidential) is required to decrypt the communication
- Alice and Bob no longer have to have *a priori* shared a secret key

Problem? YES. *How do we know if Bob's key is really Bob's?*

# Public Key Cryptography

- Each key pair consists of a public and private component: $k^+$ (public key), $k^-$ (private key)

$$D_{k^-}(E_{k^+}(m)) = m$$

- Public keys are distributed (typically) through public key certificates
  - Anyone can communicate secretly with you ***if they have your certificate***

# Encryption using private key

- Encryption and Decryption

  $E_{k-}(M)$ : ciphertext = plaintext$^d$ mod n

  $D_{k+}$(ciphertext) : plaintext = ciphertext$^e$ mod n

- E.g.,
  - $E(\{3,33\},4) = 4^3 \bmod 33 = 64 \bmod 33 = 31$
  - $D(\{7,33\},31) = 31^7 \bmod 33 = 27{,}512{,}614{,}111 \bmod 33 = 4$
- Q: Why encrypt with private key?

# Digital Signatures

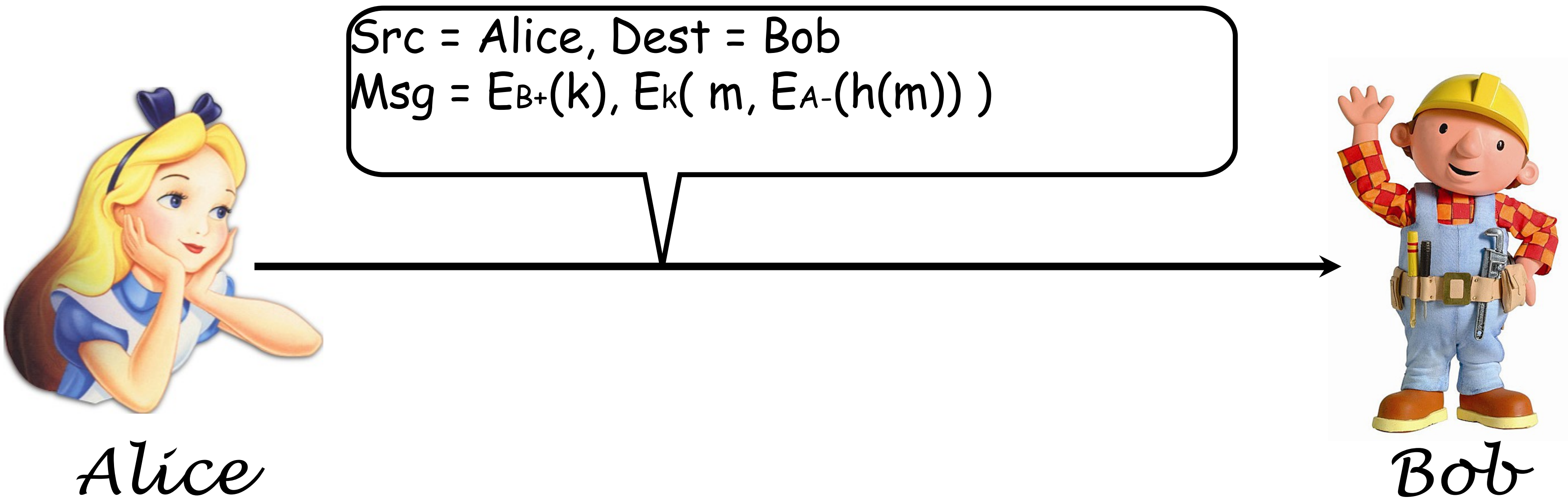
- A digital signature serves the same purpose as a real signature.
  - It is a mark that only sender can make
  - Other people can easily recognize it as belonging to the sender
- Digital signatures must be:
  - **Unforgeable**: If Alice signs message M with signature S, it is impossible for someone else to produce the pair (M, S).
  - **Authentic**: If Bob receives the pair (M, S) and knows Alice's public key, he can check ("verify") that the signature is really from Alice

# How can Alice *sign* a digital document?

- Digital document: M
- Since RSA is slow, hash M to compute digest: $m = h(M)$
- Signature: $Sig(M) = E_{k-}(m) = m^d \bmod n$
  - Since only Alice knows $k^-$, only she can create the signature
- To verify: Verify(M,Sig(M))
  - Bob computes h(M) and compares it with $D_{k+}(Sig(M))$
  - Bob can compute $D_{k+}(Sig(M))$ since he knows $k^+$ (Alice's public key)
  - If and only if they match, the signature is verified (otherwise, verification fails)

# Putting it all together

Define m = “Network security is fun!”




($A^+$, $A^-$) is Alice’s long-term public-private key pair.
($B^+$,$B^-$) is Bob’s long-term public-private key pair.
k is the session key; sometimes called the **ephemeral key**.

# Birthday Attack and Signatures

- Since signatures depend on hash functions, they also depend on the hash function's collision resistance
- Don't use MD5 or SHA1, and start moving away from SHA2

Dear Anthony,

{This letter is | I am writing} to introduce {you to | to you} {Mr. | --} Alfred {P. | --} Barton, the {new | newly appointed} {chief | senior} jewellery buyer for {our | the} Northern {European | Europe} {area | division}. He {will take | has taken} over {the | --} responsibility for {all | the whole of} our interests in {watches and jewellery | jewellery and watches} in the {area | region}. Please {afford | give} him {every | all the} help he {may need | needs} to {seek out | find} the most {modern | up to date} lines for the {top | high} end of the market. He is {empowered | authorized} to receive on our behalf {samples | specimens} of the {latest | newest} {watch and jewellery | jewellery and watch} products, {up | subject} to a {limit | maximum} of ten thousand dollars. He will {carry | hold} a signed copy of this {letter | document} as proof of identity. An order with his signature, which is {appended | attached} {authorizes | allows} you to charge the cost to this company at the {above | head office} address. We {fully | --} expect that our {level | volume} of orders will increase in the {following | next} year and {trust | hope} that the new appointment will {be | prove} {advantageous | an advantage} to both our companies.

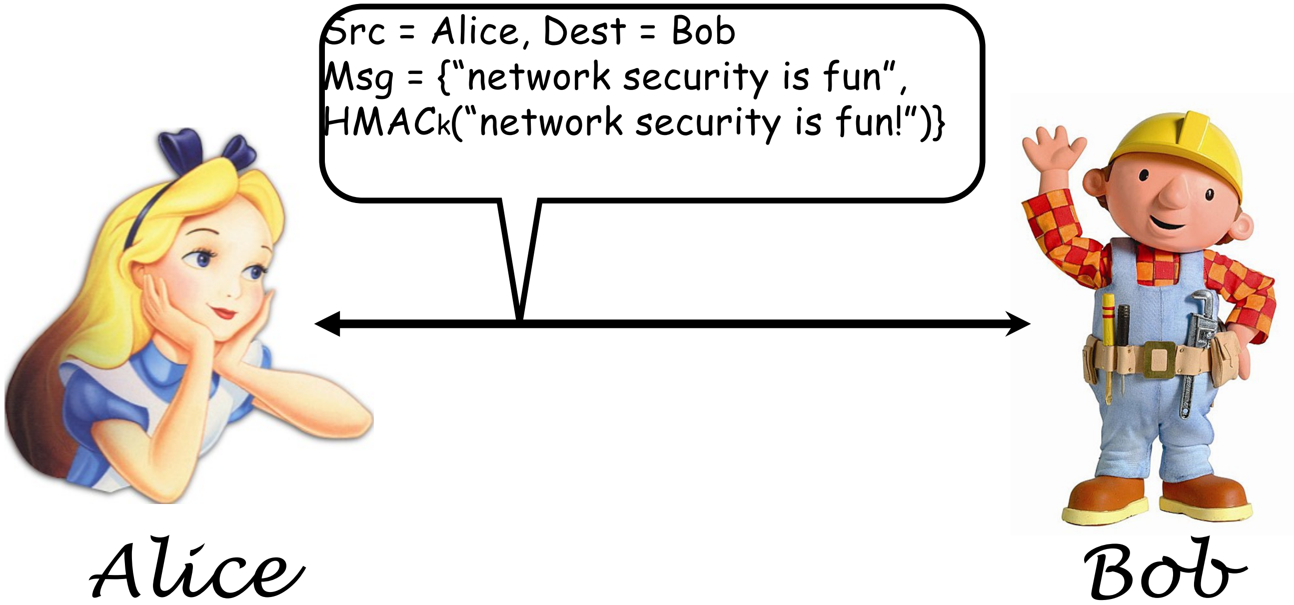
**Figure 11.7 A Letter in $2^{37}$ Variations**

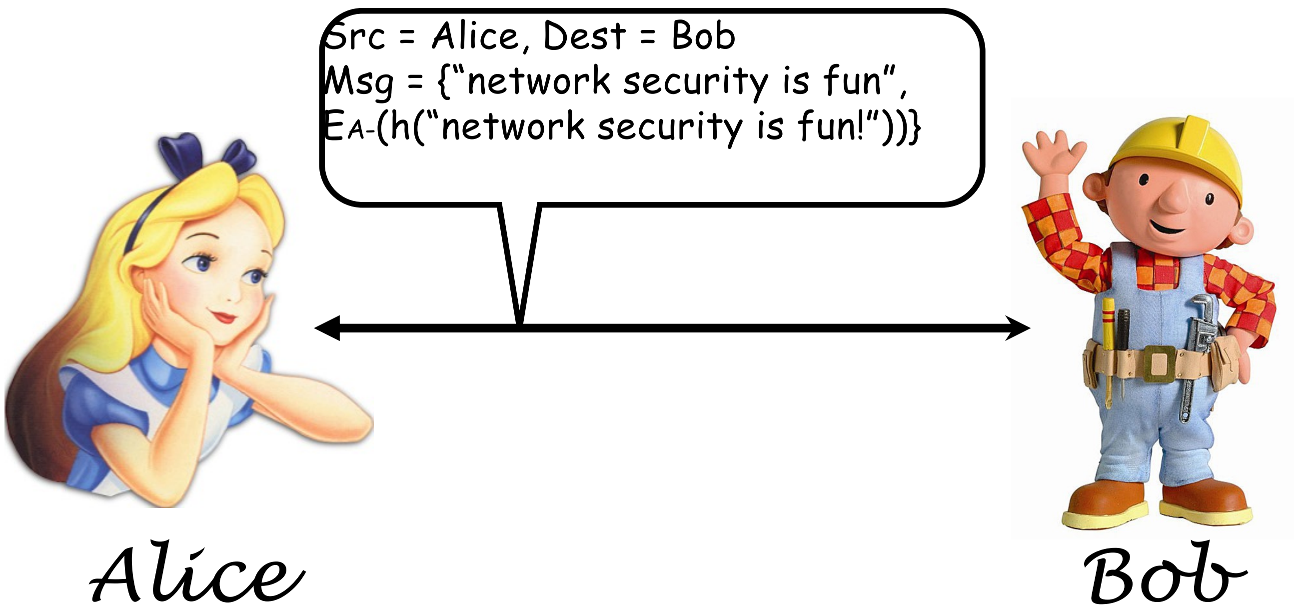
(from Stallings, Crypto and Net Security)

# Properties of a Digital Signature

- **No forgery possible:** No one can forge a message that is purportedly from Alice
- **Authenticity check:** If you get a signed message you should be able to verify that it's really from Alice
- **No alteration/Integrity:** No party can undetectably alter a signed message
- Provides authentication, integrity, and **non-repudiation** (cannot deny having signed a signed message)

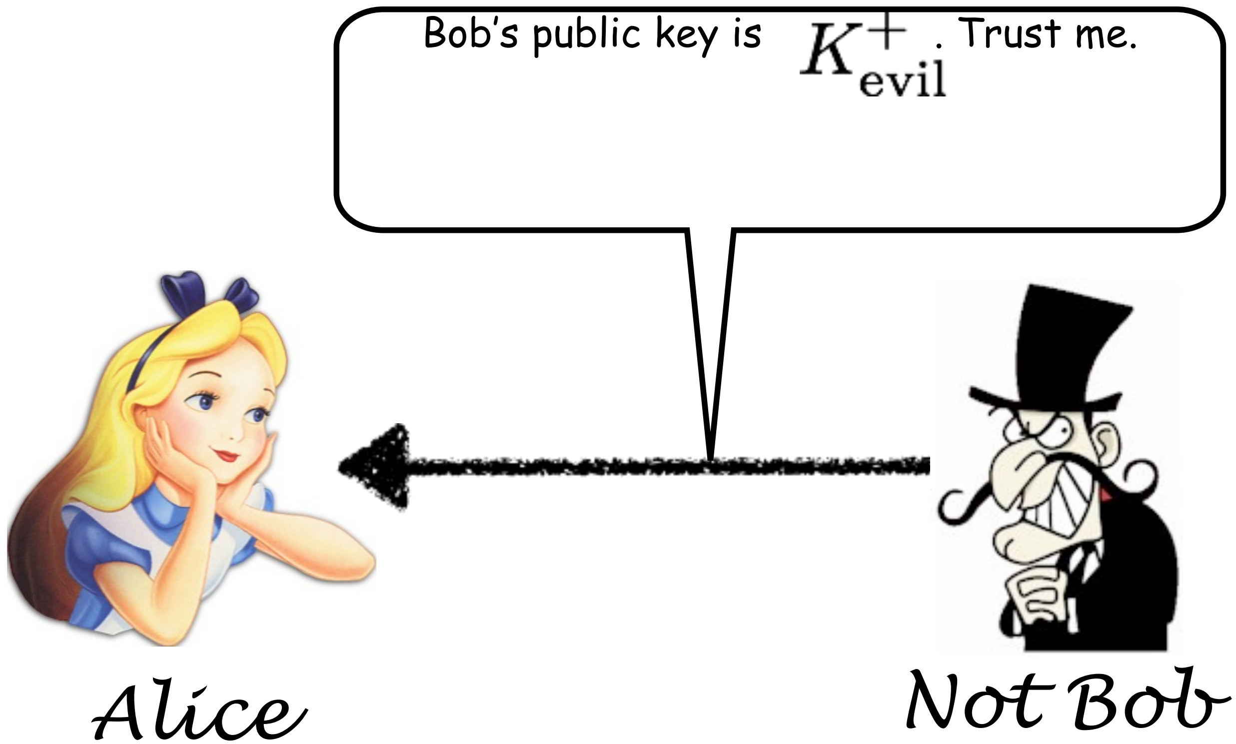
# Non-Repudiation



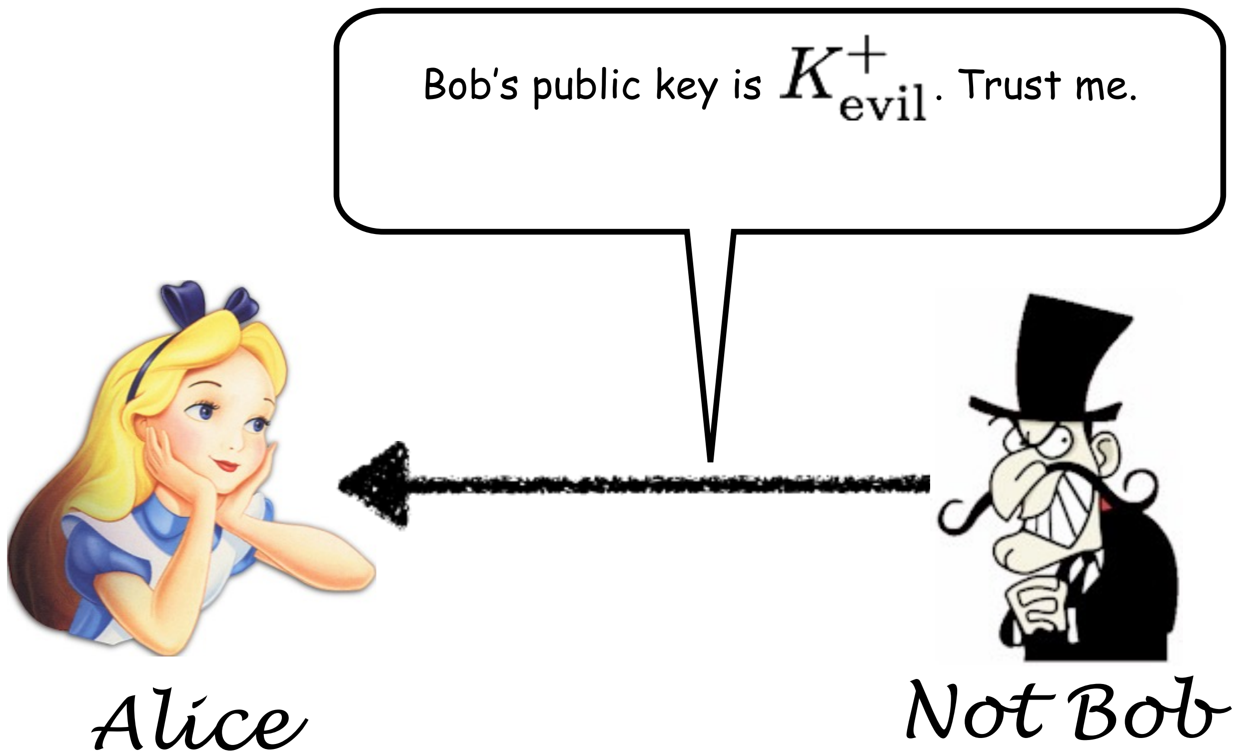




Which of these offer non-repudiation?

# But how do we *verify* we're using the correct public key?

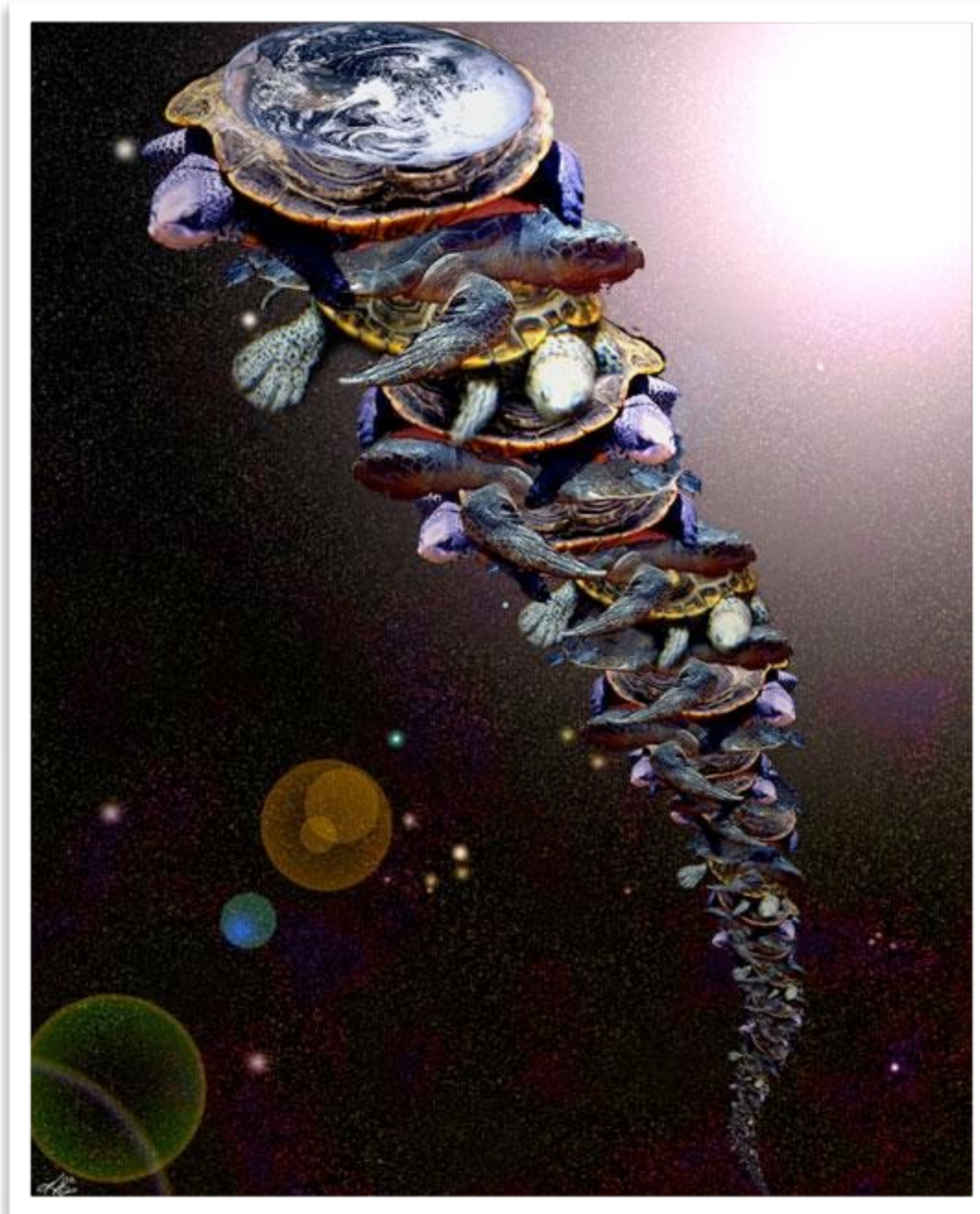


# But how do we *verify* we're using the correct public key?



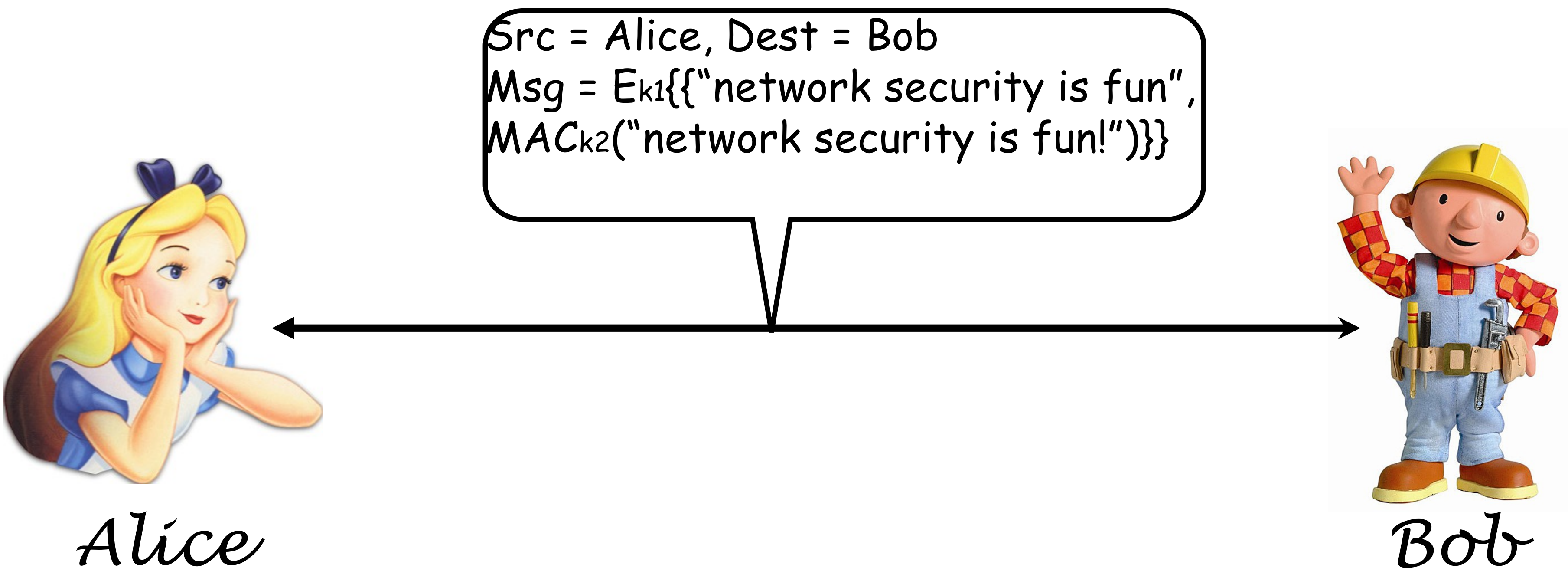
Short answer: We can't.

It's turtles all the way down.



# Authentication, Part I: Sharing a Private (Symmetric) Key

# Encryption and Message Authenticity



# Key Distribution

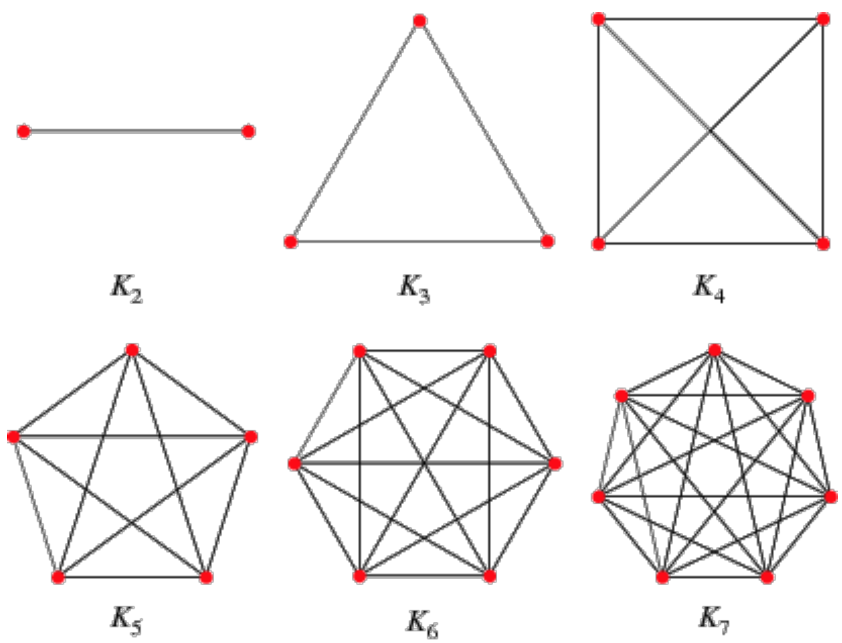
- Suppose Alice has an channel for communicating with Bob.
- Alice and Bob wish to use this channel to established a shared secret.
- However, Eve is able to learn everything sent over the channel.
- If Alice and Bob have no other channel to use, can they establish a shared secret that Eve does not know?

# Key Distribution

- Secure key distribution without asymmetric cryptography is difficult
- Simple approach: send key through an out-of-band channel

# Key Distribution

- Pairwise key distribution requires $\binom{N}{2}$ plastic cups



# Key Distribution and Key Agreement

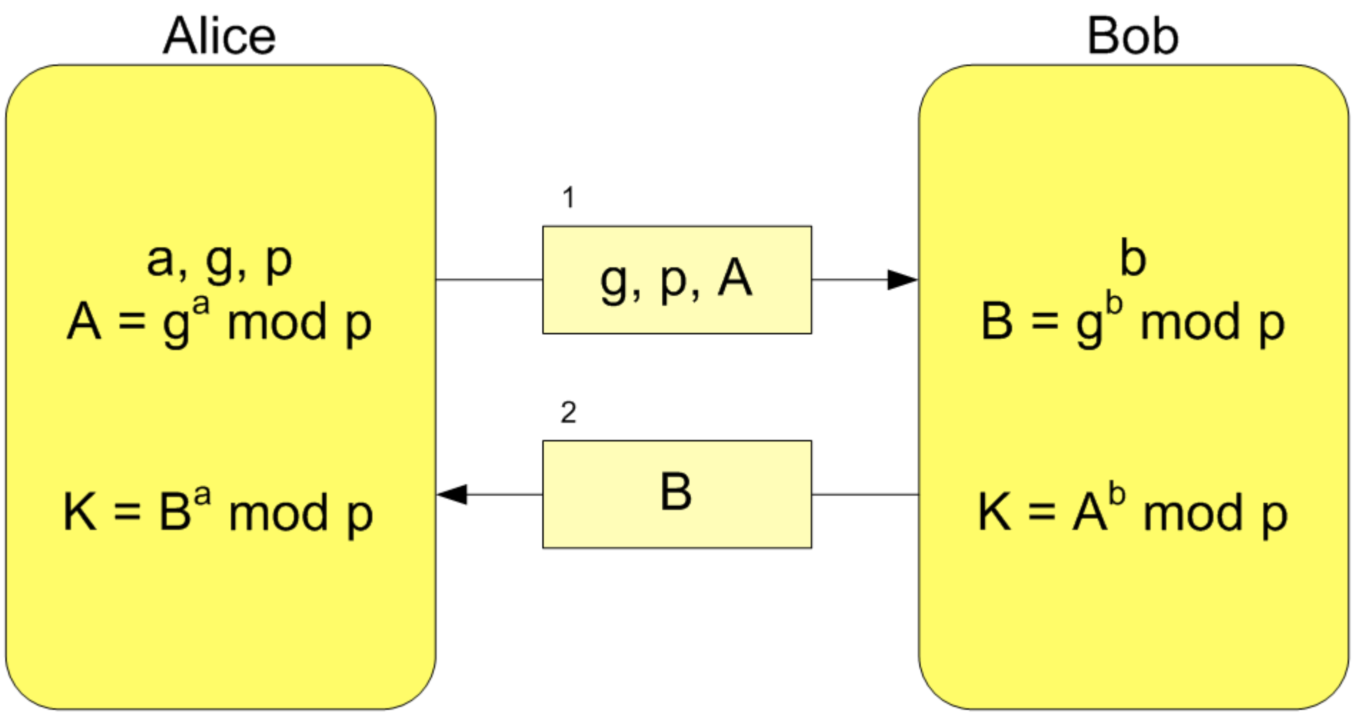
- **Key Distribution** is the process where we assign and transfer keys to a participant
  - Out of band (e.g., passwords, simple)
  - During authentication (e.g., Kerberos)
- **Key Agreement** is the process whereby two or more parties negotiate a key

# Diffie-Hellman (DH) Key Agreement

- The DH paper started the modern age of cryptography, and indirectly the security community
  - Negotiate a secret over an insecure media
  - E.g., "in the clear" (seems impossible)
  - Idea: participants exchange intractable puzzles that can be solved easily with additional information
- Mathematics are very deep
  - Use the hardness of computing discrete logarithms in finite field to make secure

# Diffie-Hellman (DH) Key Agreement

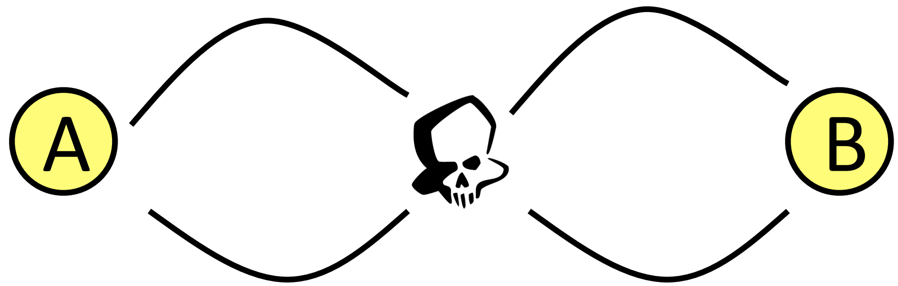
- Proposed by Whitfield Diffie and Martin Hellman in 1976
- g=base, p=prime (>512 bits), a=Alice's secret, b=Bob's secret
  - g is a *primitive root* of p, and g < p ; p and g are *publicly known*
- Eve cannot compute K without knowing either a or b (neither of which is transmitted), even if she (passively) intercepts all communication!




$K = A^b \bmod p = (g^a \bmod p)^b \bmod p = g^{ab} \bmod p = (g^b \bmod p)^a \bmod p = B^a \bmod p$

# Attacks on Diffie-Hellman

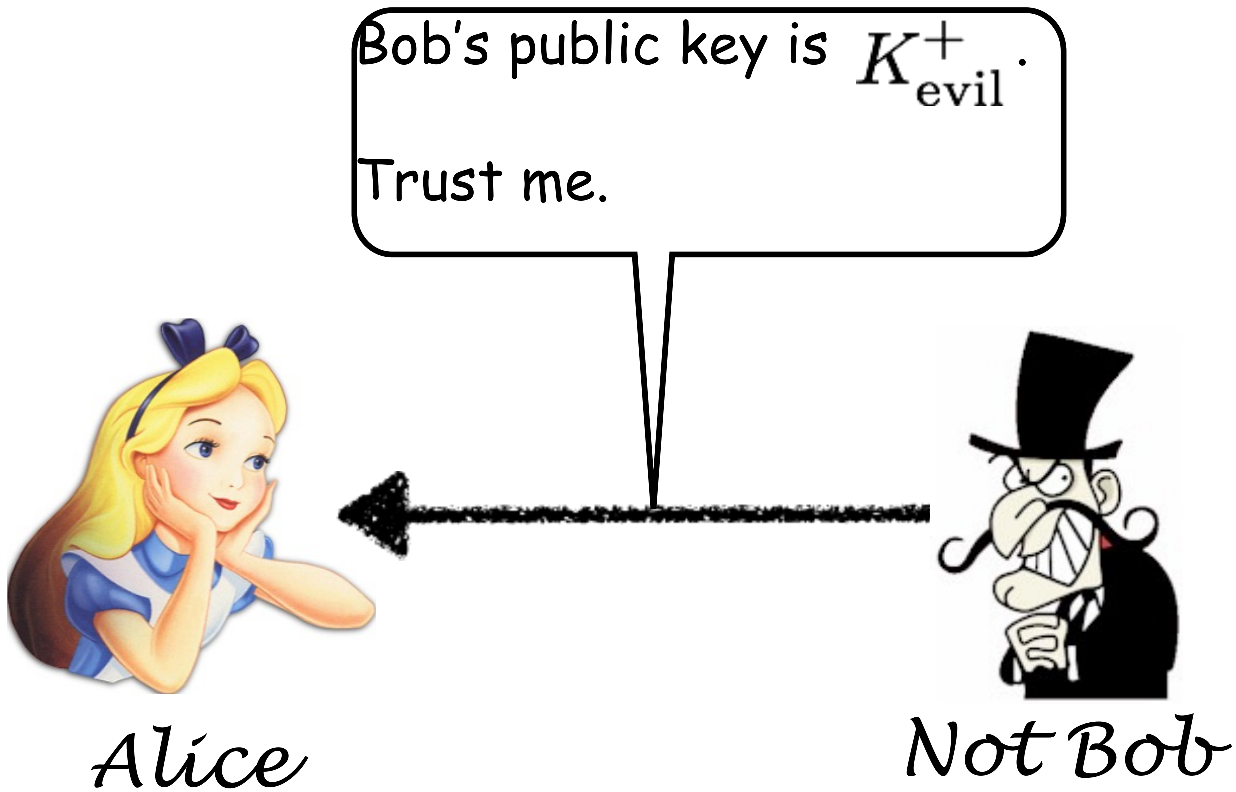
- Subject to **Man-in-the-Middle** (MitM) attack
  - You really don't know anything about who you have exchanged keys with




  - Alice and Bob think they are talking directly to each other, but Mallory is actually performing two separate exchanges
- Fix: Authenticated DH exchange
  - The parties sign the exchanges (more or less)
  - Requires pre-shared knowledge or trusted third party

# Authentication Part II: Public Key Distribution

# How do we *verify* we’re using the correct public key?

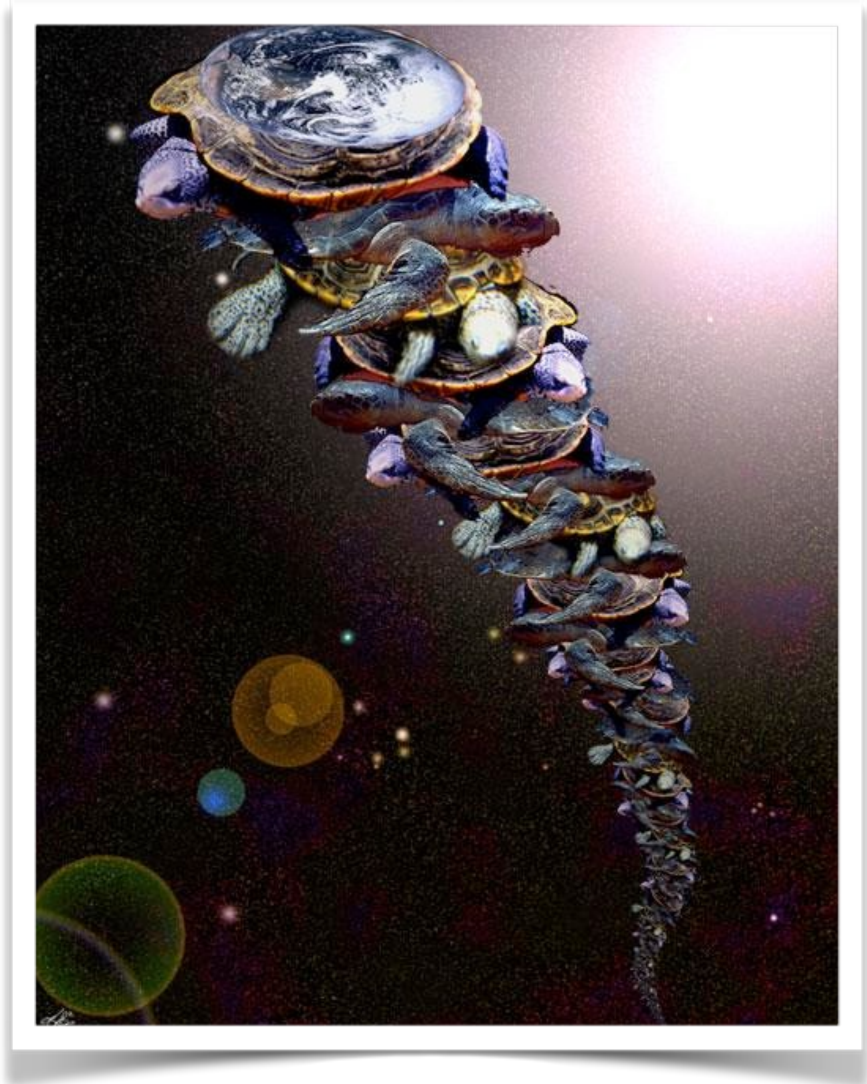


# Why not just use a database?

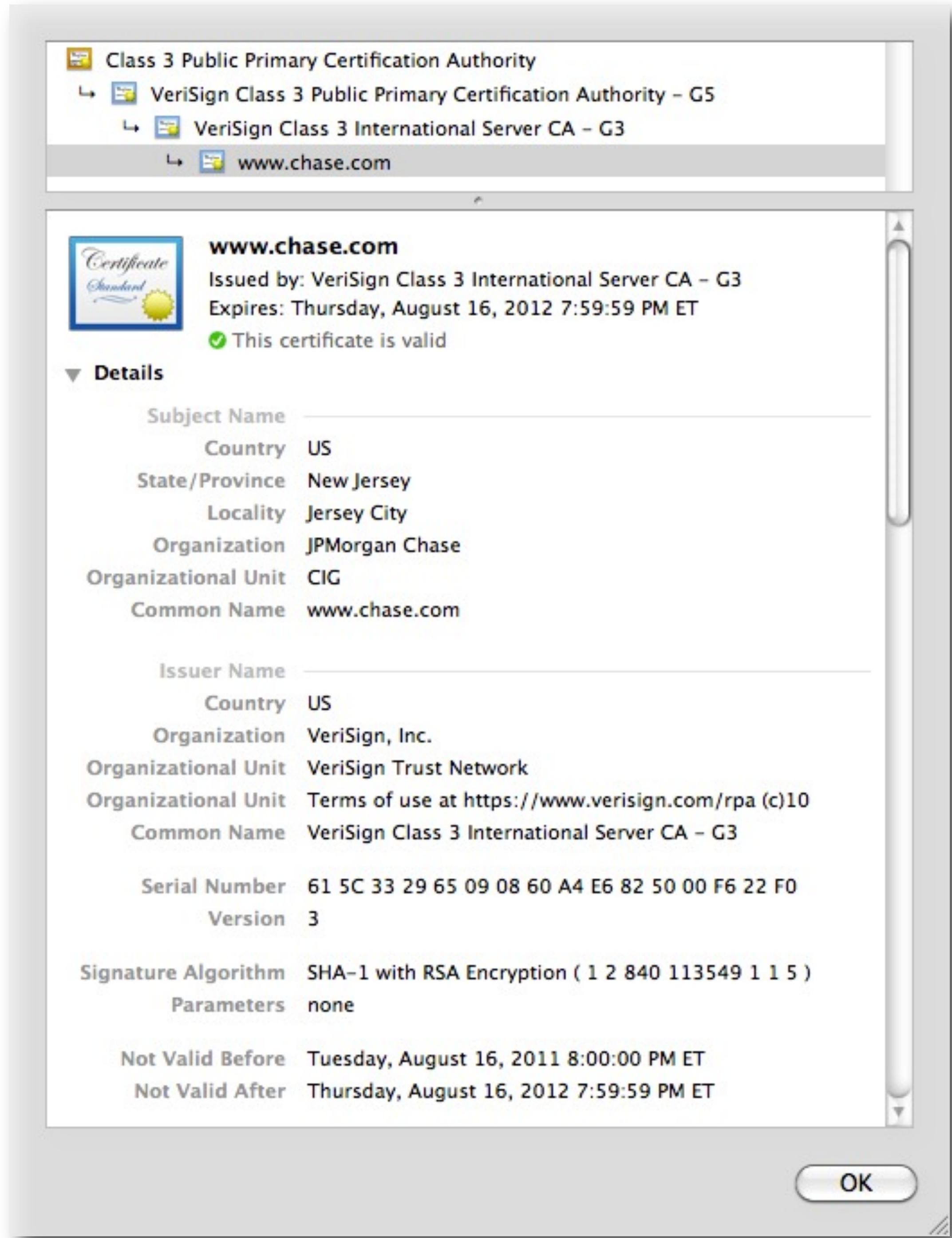
- Every user has his/her own public key and private key.
- Public keys are all published in a database.
- Alice gets Bob's public key from the database
- Alice encrypts the message and sends it to Bob using Bob's public key.
- Bob decrypts it using his private key.
- What's the problem with this approach?

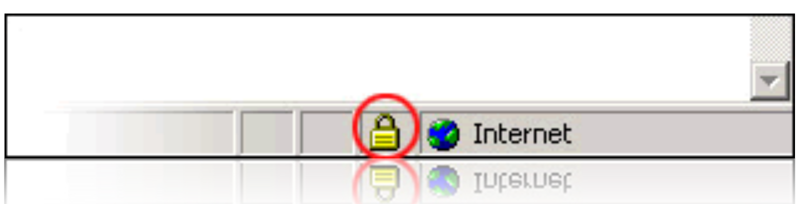
# Solving the Turtles Problem

- We need a **trust anchor**
  - there must be someone with authority
  - requires *a priori* trust
- Solution: form a trust hierarchy
  - "I believe **X** because..."
  - "**Y** vouches for **X** and..."
  - "**Z** vouches for **Y** and..."
  - "I implicitly trust **Z**."



# Browser Certificate



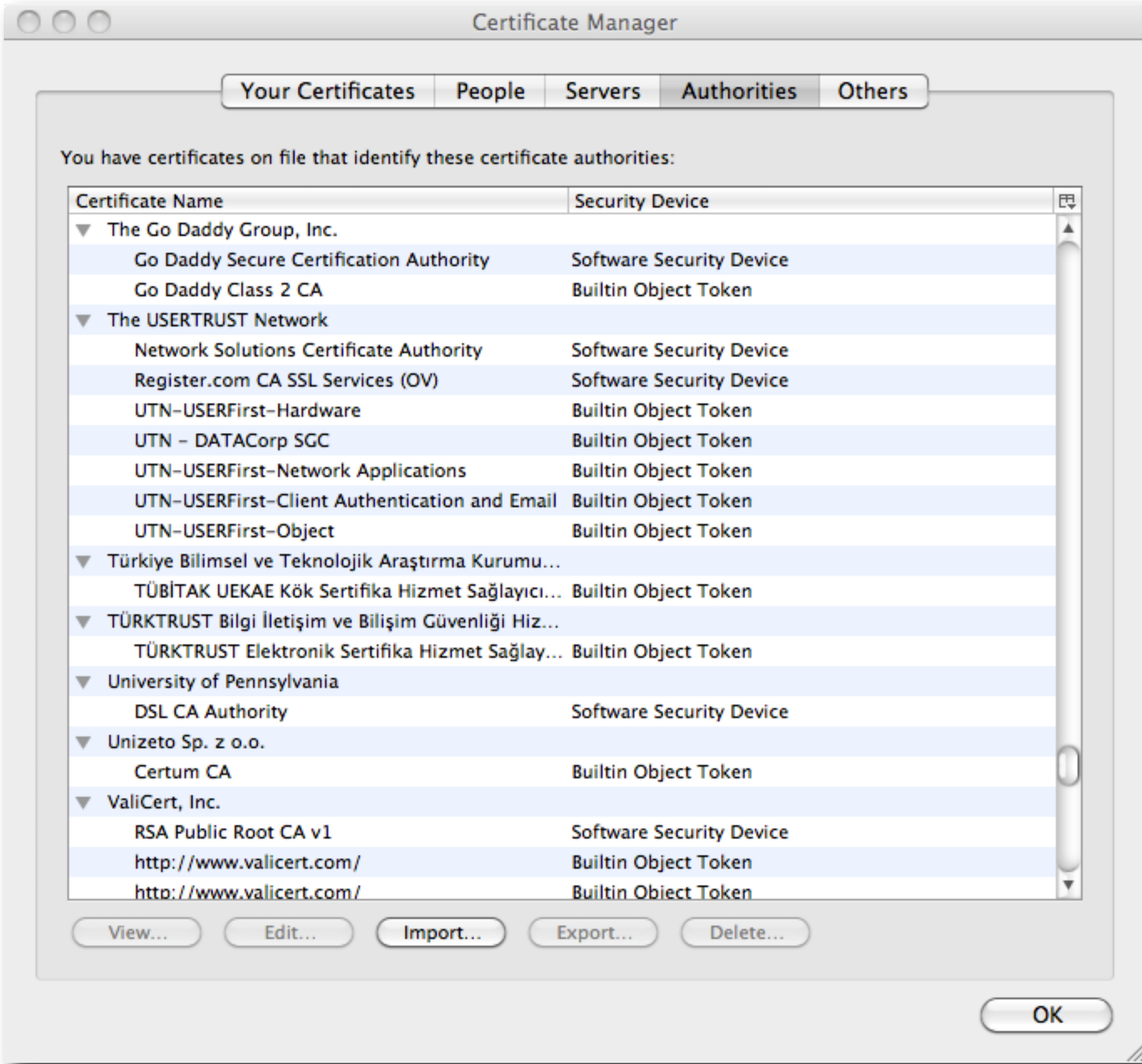



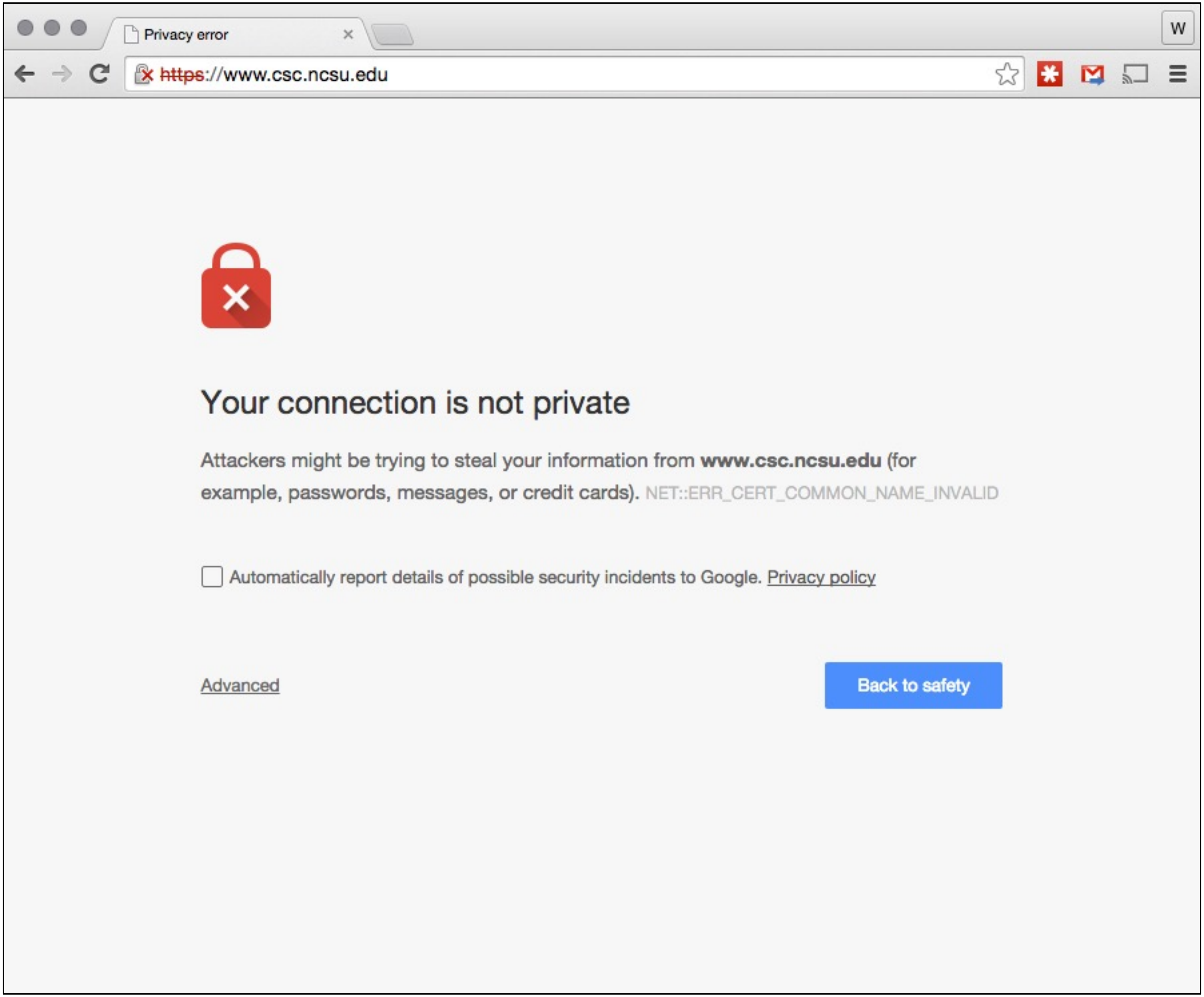
# What's a certificate?

- A certificate …
  - … makes an association between an identity and a private key
  - … contains public key information {e,n}
  - … has a validity period
  - … is signed by some *certificate authority* (CA)
  - … identity may have been vetted by a *registration authority* (RA)
- People trust CA (e.g., Verisign) to vet identity

# Why do I trust the certificate?

- A collections of “root” CA certificates
  - … baked into your browser
  - … vetted by the browser manufacturer
  - … <u>supposedly</u> closely guarded
- Root certificates used to validate certificate
  - Vouches for certificate’s authenticity


Certificate Manager
Your Certificates
People
Servers
Authorities
Others
You have certificates on file that identify these certificate authorities:
Certificate Name
Security Device
The Go Daddy Group, Inc.
Go Daddy Secure Certification Authority
Software Security Device
Go Daddy Class 2 CA
Builtin Object Token
The USERTRUST Network
Network Solutions Certificate Authority
Software Security Device
Register.com CA SSL Services (OV)
Software Security Device
UTN-USERFirst-Hardware
Builtin Object Token
UTN - DATACorp SGC
Builtin Object Token
UTN-USERFirst-Network Applications
Builtin Object Token
UTN-USERFirst-Client Authentication and Email
Builtin Object Token
UTN-USERFirst-Object
Builtin Object Token
Türkiye Bilimsel ve Teknolojik Araştırma Kurumu...
TÜBİTAK UEKAE Kök Sertifika Hizmet Sağlayıcı...
Builtin Object Token
TÜRKTRUST Bilgi İletişim ve Bilişim Güvenliği Hiz...
TÜRKTRUST Elektronik Sertifika Hizmet Sağlay...
Builtin Object Token
University of Pennsylvania
DSL CA Authority
Software Security Device
Unizeto Sp. z o.o.
Certum CA
Builtin Object Token
ValiCert, Inc.
RSA Public Root CA v1
Software Security Device
http://www.valicert.com/
Builtin Object Token
http://www.valicert.com/
Builtin Object Token
View...
Edit...
Import...
Export...
Delete...
OK

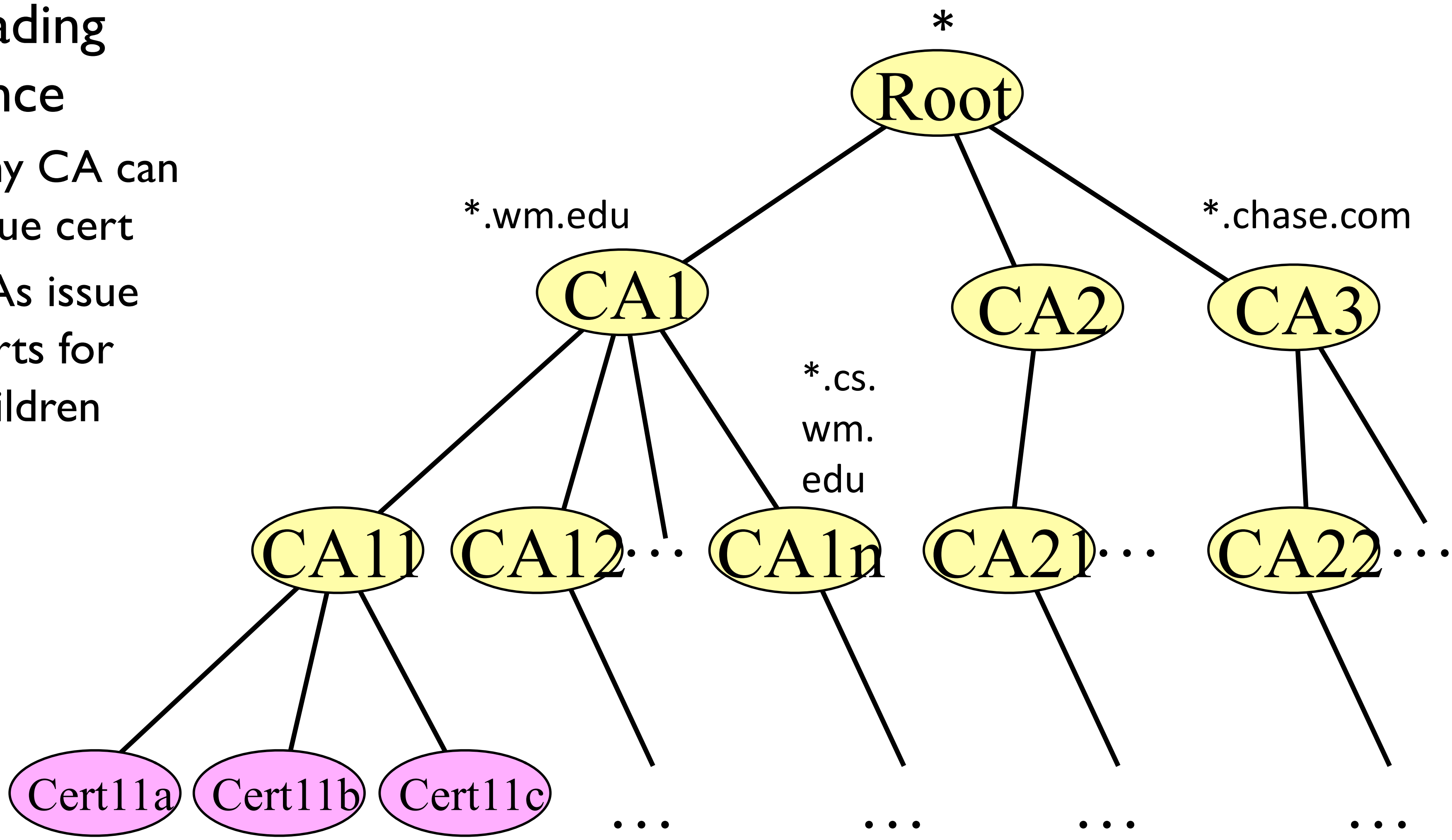

Privacy error
W
https://www.csc.ncsu.edu
Your connection is not private
Attackers might be trying to steal your information from www.csc.ncsu.edu (for example, passwords, messages, or credit cards). NET::ERR_CERT_COMMON_NAME_INVALID
Automatically report details of possible security incidents to Google. Privacy policy
Advanced
Back to safety

# Public Key Infrastructure

- Hierarchy of keys used to authenticate certificates
- Requires a **root of trust** (i.e., a **trust anchor**)

# What is a PKI?

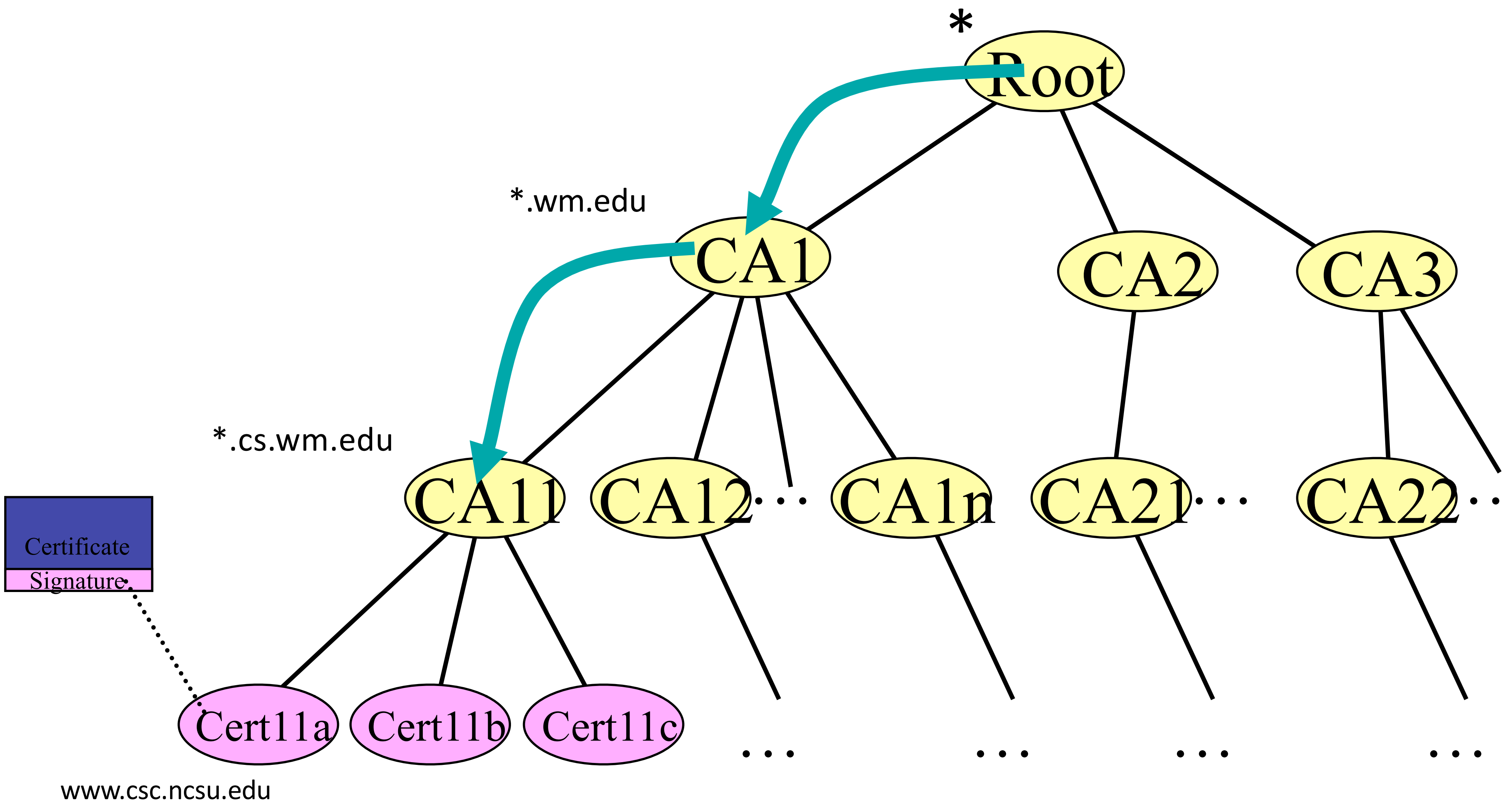
- Rooted tree of CAs
- Cascading issuance
  - Any CA can issue cert
  - CAs issue certs for children



# Obtaining a Certificate

1. Alice has some identity document $A^{ID}$ and generates a keypair ($A^-$, $A^+$)
2. $A \rightarrow CA$ : $\{A^+, A^{ID}\}$, $Sig(A^-, \{A^+, A^{ID}\})$
   - CA verifies signature -- proves Alice has $A^-$
   - CA may (and should!) also verify $A^{ID}$ offline
3. CA signs $\{A^+, A^{ID}\}$ with its private key ($CA^-$)
   - CA attests to binding between A+ and $A^{ID}$
4. $CA \rightarrow A$ : $\{A^+, A^{ID}\}$, $Sig(CA^-, \{A^+, A^{ID}\})$
   - this is the certificate; Alice can freely publish it
   - anyone who knows $CA^+$ (and can therefore validate the CA's signature) knows that CA "attested to" $\{A^+, A^{ID}\}$
   - note that CA never learns $A^-$

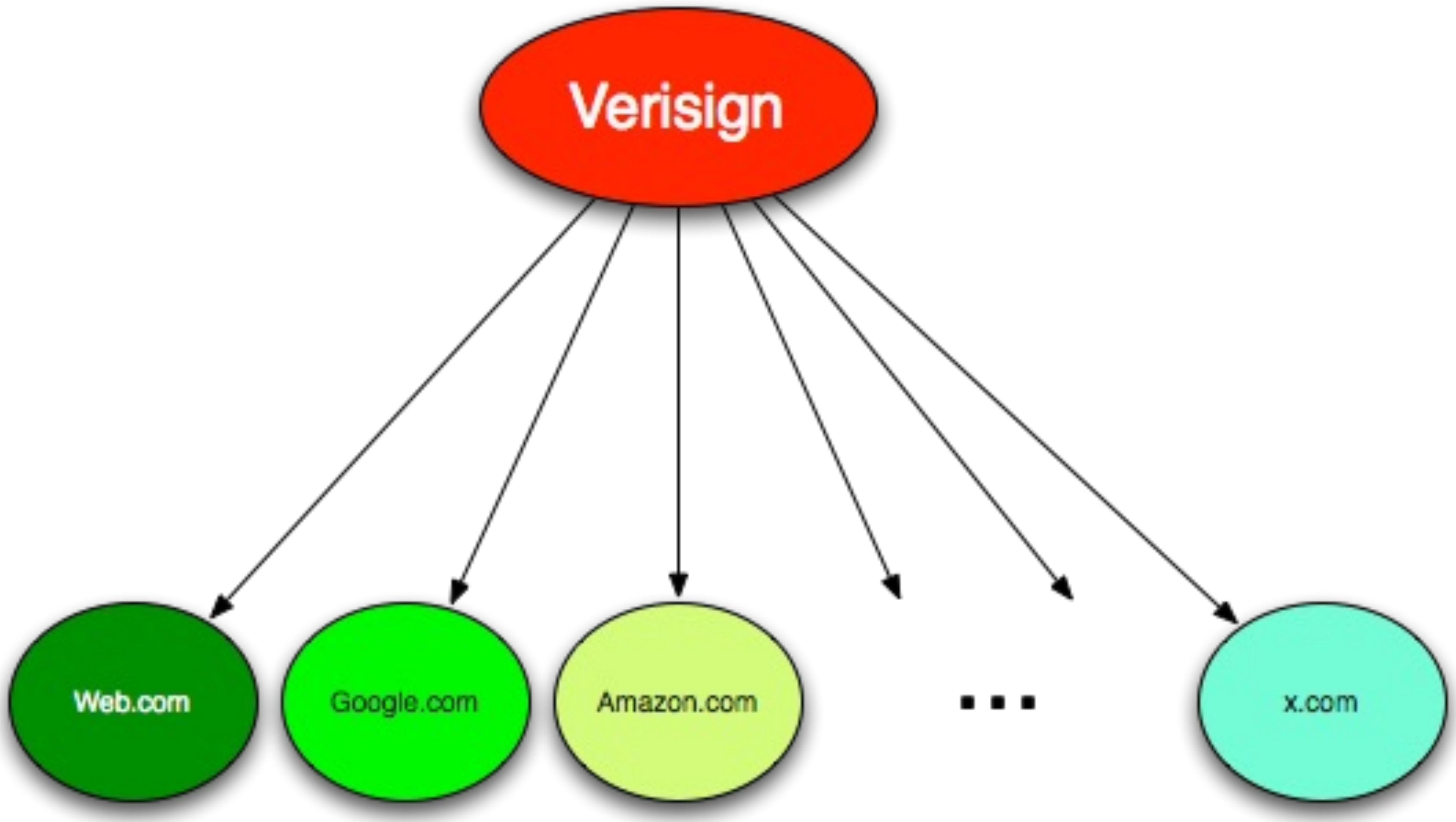
# Certificate Validation



# Certification Authorities

- Guarantee connection between public key and end entity
  - Man-in-the-Middle no longer works undetected
    - *(If you verify the identity in the certificate against peer)*
  - Guarantee authentication and non-repudiation
    - *(If a CA doesn't make a mistake)*
  - Privacy/confidentiality not an issue here
    - Only concerned with linking key to owner
- Distribute responsibility
  - Hierarchical structure
    - (Doesn't exist in practice-- no good way to restrict delegation)

# PKIs in Reality

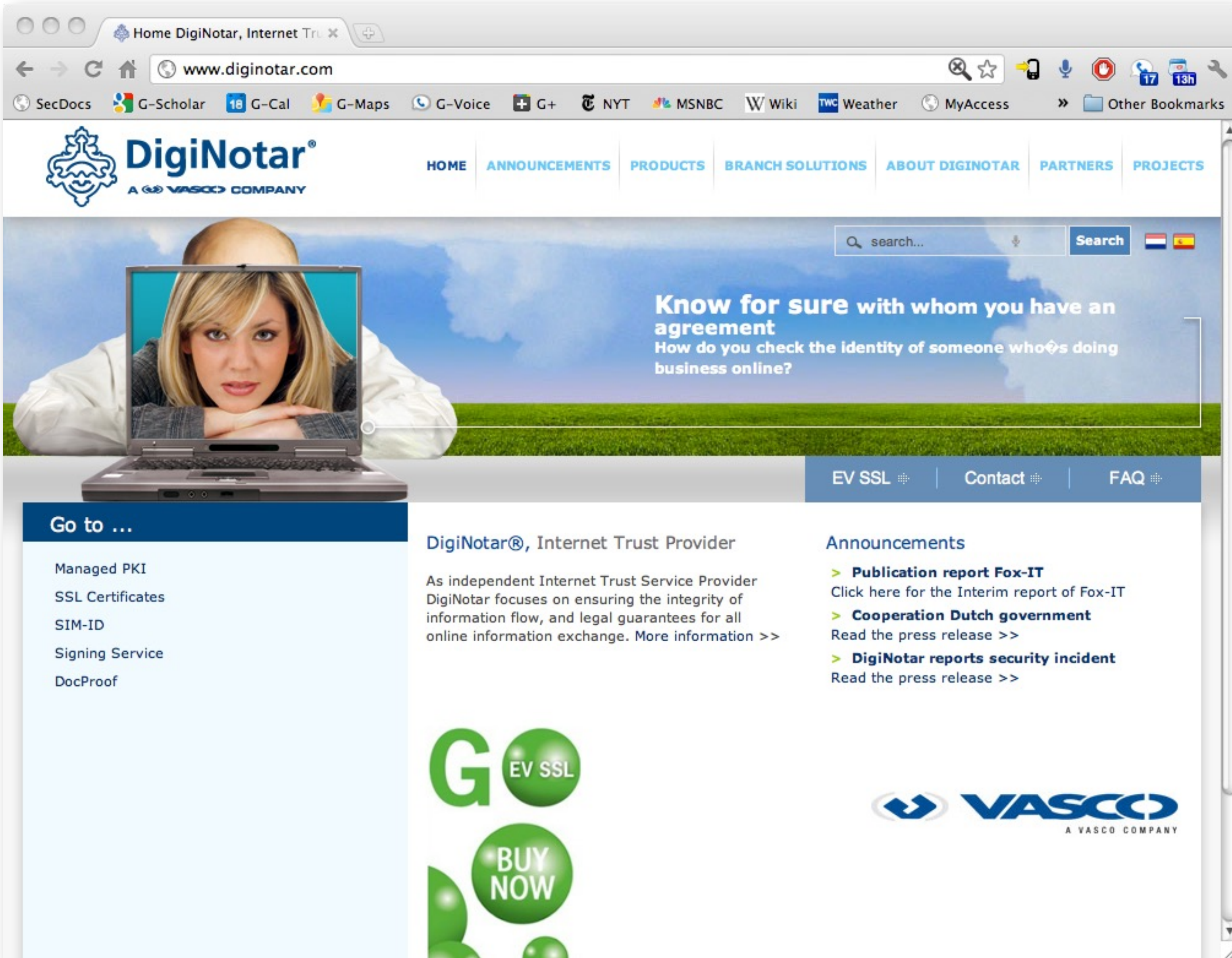


# PKI and Revocation

- Certificate may be revoked before expiration
  - Lost private key
  - Compromised
  - Owner no longer authorized
- Revocation is hard …
  - Verifiers need to check revocation state
    - Loses the advantage of off-line verification
  - Revocation state must be authenticated

- Any CA may sign any certificate
- Browser weighs all root CAs equally
- *Q: Is this problematic?*

# The DigiNotar Incident

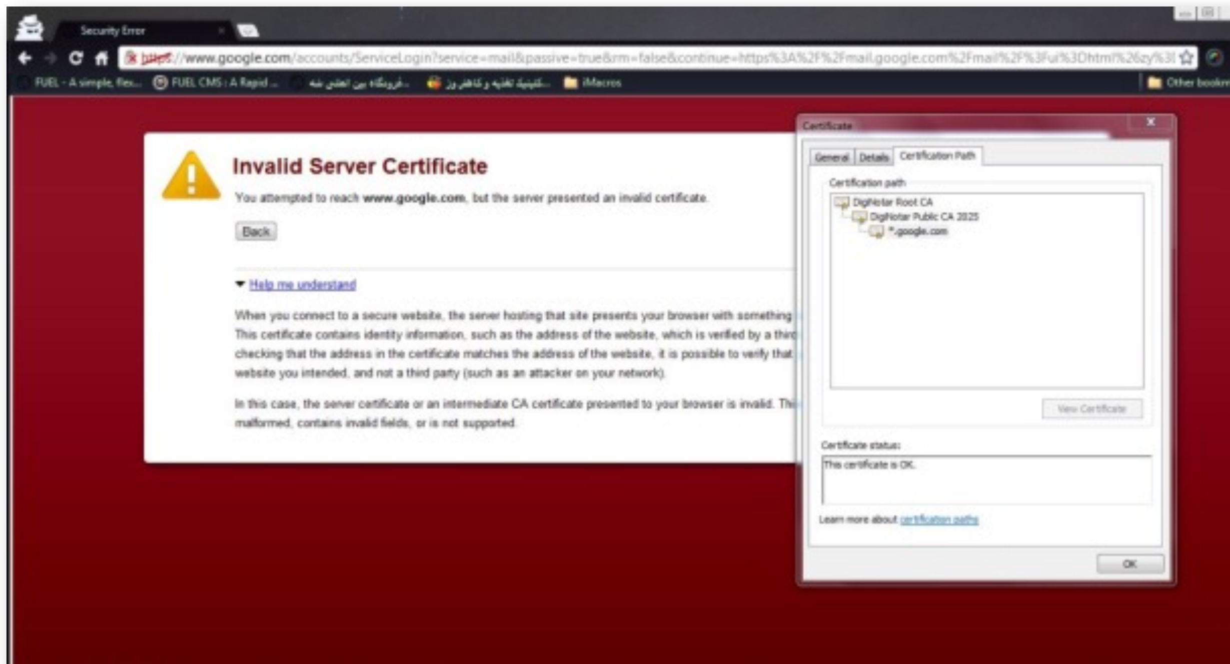


# DigiNotar Incident

- DigiNotar is a CA based in the Netherlands that is (well, was) trusted by most OSes and browsers
- July 2011: Issued fake certificate for gmail.com to site in Iran that ran MitM attack...
- ... this fooled most browsers, but...

# DigiNotar Incident

- As added security measure, Google Chrome hardcodes fingerprint of Google's certificate
- Since DigiNotar didn't issue Google's true certificate, this caused an error message in Chrome



# Meta-Issue: How much should we trust CAs?

(Because right now, we trust them a lot.)

# 10 Risks of PKI

Carl Ellison and Bruce Schneier

- PKI, like many security technologies, claimed to be a panacea
- It was intended to solve a very hard problem: build trust on a global level

# Risk 1: Who do we trust, and for what?

- Argument: CA is not inherently trustworthy
  - Why do/should you trust a CA?
  - Risk in the hands of the certificate holder
- Counter-Argument: Incentives
  - Any CA caught misbehaving is going to be out of business tomorrow
  - Risk held by everybody, which is what you want
    - Everyone has reason to be diligent

# Risk 2: Who is using my key?

- How do you protect your certificate?
- Is your computer/network completely secure?
- Who is responsible if your key is compromised?

# Risk 3: How secure is the verifier?

- What happens if attacker is able to insert his public root CA key to the verifier's list of trusted CAs?
- More generally, what are the consequences if the verifier is compromised?
- Q: What's in your browser?
  - E.g., Superfish

# Risk 4: Which John Robinson is he?

- Argument: identity in PKI is too loosely defined
  - No standards for getting credential
  - No publicly known unique identifiers for people
  - So, how do you tell people apart
- Counter-Argument: due diligence
  - Only use certificates in well known circumstances
  - When in doubt, use other channels to help

# Risk 5:
# Is the CA an authority?

- Argument: there are things in certificates that claim authenticity and authorization of which they have no dominion
  - DNS, attributes -- the CA is not the arbiter of these things

# Risk 8: How did the CA identify the certificate holder?

- How well do CAs really authenticate the person requesting the certificate?
- What are the potential consequences?

# Risk 9: How secure are the certificate practices?

- What happens if the CA's private key is compromised?
- Are certificate revocation lists (CRLs) used?
- What is an appropriate certificate lifetime? [*This is both a security question and an MBA question*]

# Key Management Summary

- Key management is HARD
- PKI is not a panacea
- Devil is in the details