CIS 4930: Secure IoT

Lecture 5

Prof. Kaushal Kafle

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

Class Notes

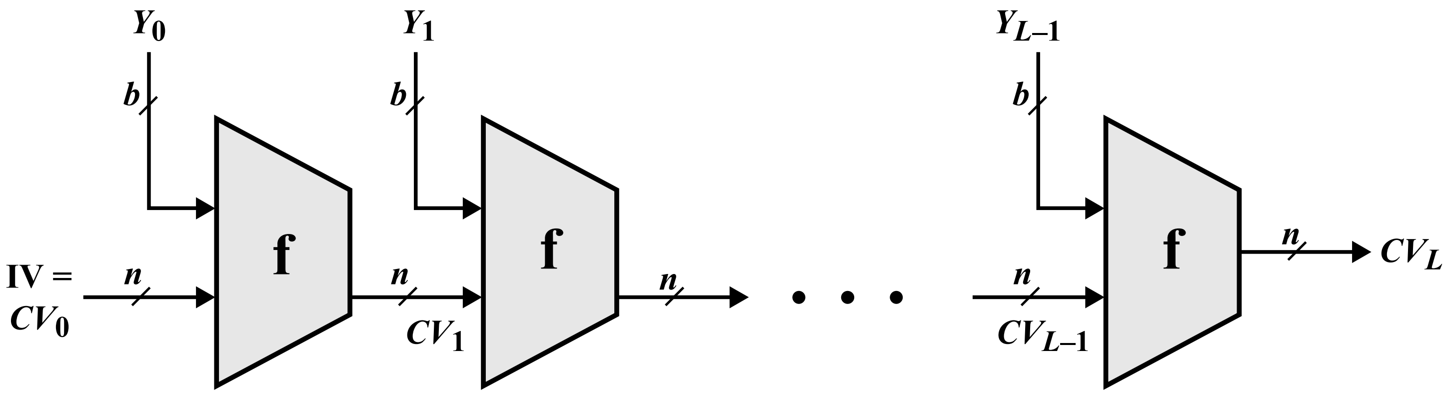
- Required to submit the finalized project for your team so meet with me **asap** to do that!
- Teams that have met -> submit the finalized proposal and begin work!

- **Midterm date**: 10/15

From Last Class

- Message Authentication Codes (MAC):
  - Generate and send a value computed using the original message and a secret key
  - Provides authenticity and integrity
- Hash functions:
  - One-way function to generate fixed-length *hash (or digest)*
  - One of the use-cases: Generating MACs!

General Structure of Hash



IV	=	Initial value	L	=	number of input blocks
CV_i	=	chaining variable	n	=	length of hash code
Y_i	=	ith input block	b	=	length of input block
f	=	compression algorithm			

(from Stallings, Crypto and Net Security)

Message Extension Attack

- Why is $MAC_k(M) = H(k|M)$ bad?
- How can Eve append M’ to M?
  - Goal: compute H(k|M|M’) without knowing k
- Solution: Use H(k|M) as IV for next f iteration in H()

A Better MAC

- Objectives
  - Use available hash functions without modification
  - Easily replace embedded hash function as more secure ones are found
  - Preserve original performance of hash function
  - Easy to use

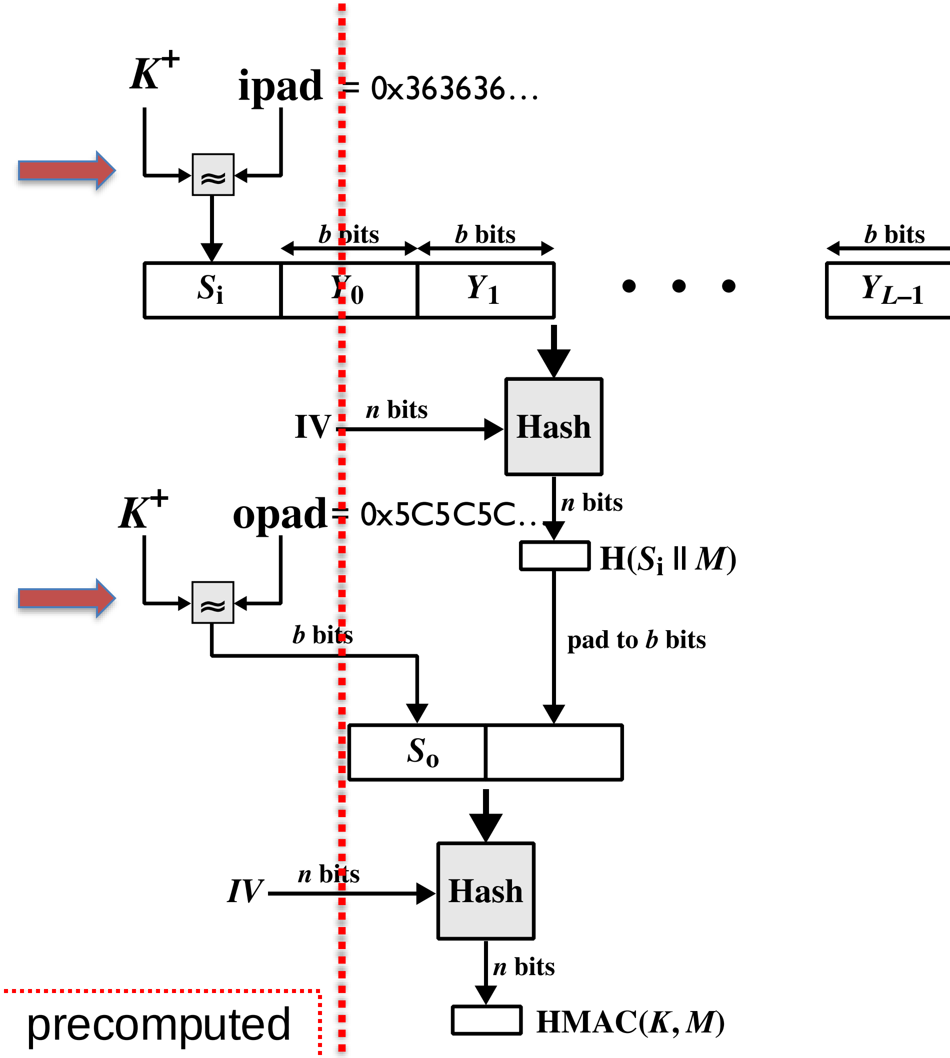
HMAC

HMAC(k, M)

H(k⊕opad || H(k⊕ipad || M))

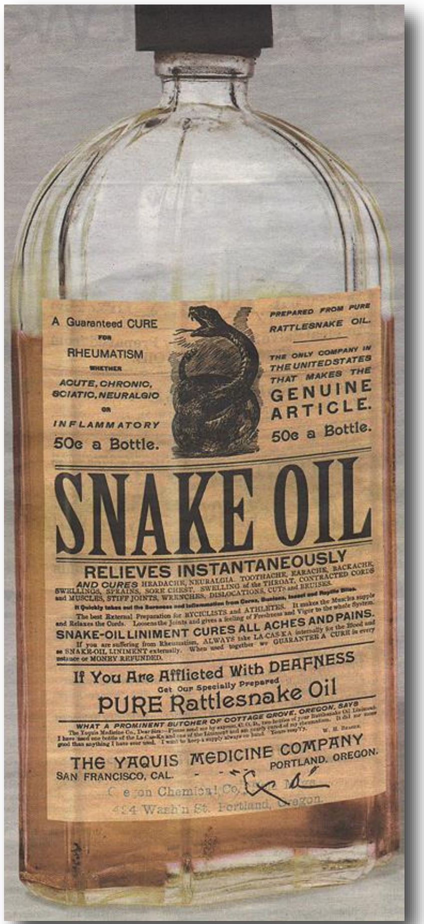
hash2 ***hash1***

- Attacker cannot extend MAC as before
  - Try it out!



(from Stallings, Crypto and Net Security)

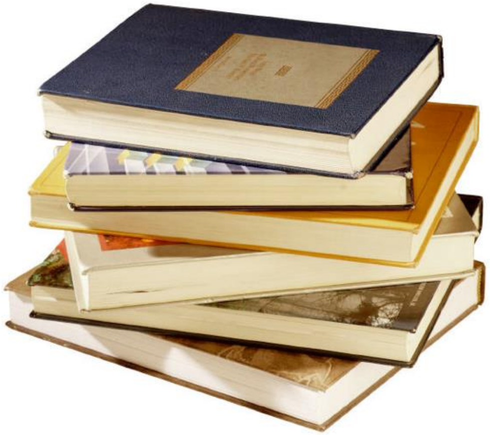
Basic truths of cryptography



- Cryptography is not frequently the source of security problems
  - Algorithms are well known and widely studied
  - Vetted through crypto community
  - Avoid any “proprietary” encryption
  - Claims of “new technology” or “perfect security” are almost assuredly **snake oil**

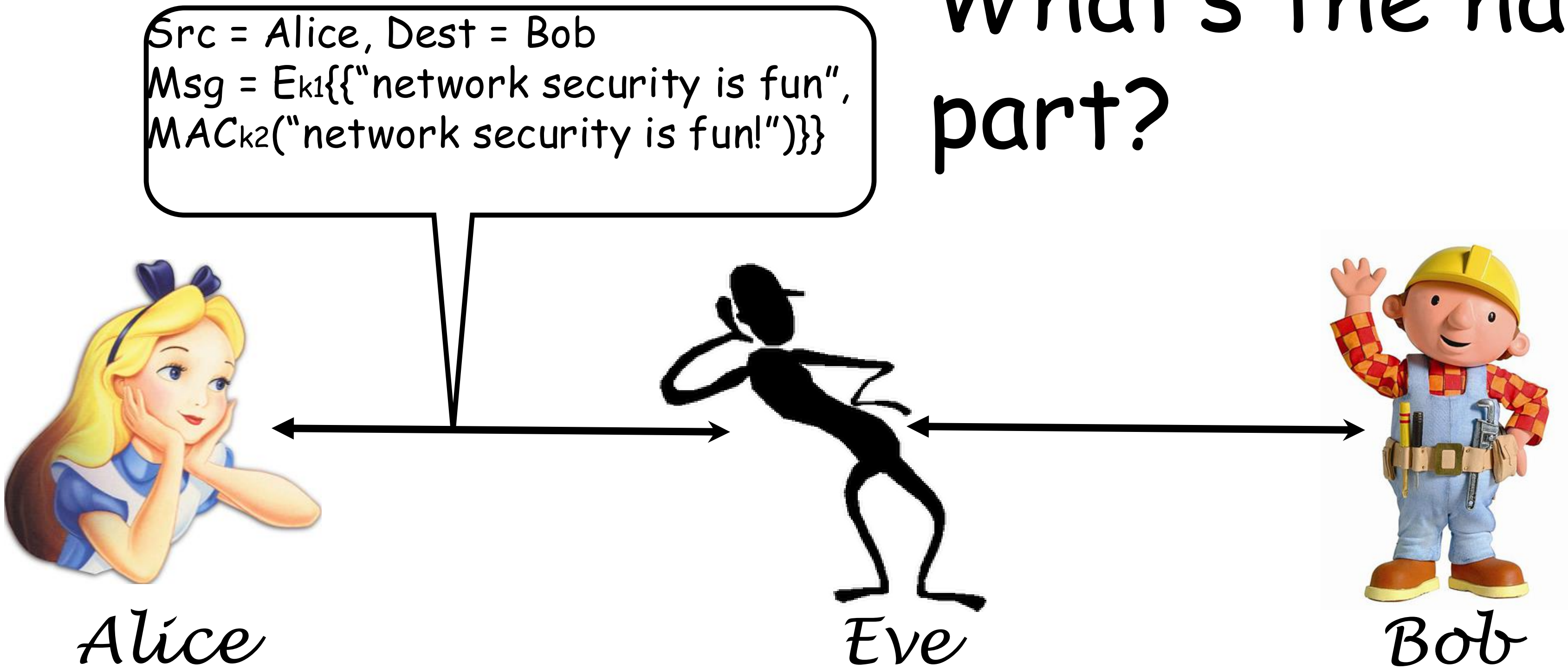
Building systems/apps with cryptography

- Use quality libraries
  - SSLeay, cryptolib, openssl
  - Find out what cryptographers think of a package before using it
- Code review like crazy
- Educate yourself on how to use library
  - Understand caveats by original designer and programmer



Encryption and Message Authenticity

What's the hard part?



Without knowing *k1*, Eve can't read Alice's message.

Without knowing *k2*, Eve can't compute a valid MAC for her forged message.

Private-key crypto is like a door lock

Why?

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

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***

RSA
(Rivest, Shamir, Adelman)

- The dominant public key algorithm
  - The algorithm itself is conceptually simple
  - Why it is secure is very deep (number theory)
  - Uses properties of exponentiation modulo a product of large primes

"A method for obtaining Digital Signatures and Public Key Cryptosystems“, Communications of the ACM, Feb. 1978.

Modular Arithmetic

- Integers $Z_n = \{0, 1, 2, ..., n-1\}$
- x mod n = remainder of x divided by n
  - 5 mod 13 = 5
  - 13 mod 5 = 3
- y is **modular inverse** of x iff **xy mod n = 1**
  - E.g. Z_{11} -> 4 is inverse of 3, 5 is inverse of 9, 7 is inverse of 8
- If **n is prime**, then Z_n has modular inverses for all integers except 0

Euler's Totient Function

- **coprime**: having no common positive factors other than 1 (also called **relatively prime**)
  - 16 and 25 are coprime
  - 6 and 27 are not coprime
- **Euler's Totient Function**: Φ(n) = number of integers less than or equal to n that are coprime with n

$$\Phi(n) = n \cdot \prod_{p|n} (1 - \frac{1}{p})$$

where product ranges over distinct primes dividing n

- If m and n are coprime, then Φ(mn) = Φ(m)Φ(n)

- If m is prime, then Φ(m) = m - 1

Euler's Totient Function

$$\Phi(n) = n \cdot \prod_{p|n} (1 - \frac{1}{p})$$

$$\Phi(18) = \Phi(3^2 \cdot 2^1) = 18(1 - \frac{1}{3})(1 - \frac{1}{2}) = 6$$

For primes and co-primes:

If m and n are coprime, then Φ(mn) = Φ(m)Φ(n)

If m is prime, then Φ(m) = m - 1

RSA Key Generation

Example:

1. Choose distinct primes p and q
2. Compute $n = pq$
3. Compute $\Phi(n) = \Phi(pq) = \Phi(p)\Phi(q) = (p-1)(q-1)$
4. Randomly choose $1 < e < \Phi(pq)$ such that e and $\Phi(pq)$ are coprime. e is the **public key exponent**
5. Compute $d = e^{-1} \bmod (\Phi(pq))$. d is the **private key exponent**

let $p=3$, $q=11$

$n=33$

$\Phi(pq)=(3-1)(11-1)=20$

let $e=7$

$ed \bmod \Phi(pq) = 1$

$7d \bmod 20 = 1$

$d = 3$

RSA Encryption/Decryption

- Public key k^+ is {e,n} and private key k^- is {d,n}
- Encryption and Decryption

$$E_{k+}(M) : \text{ciphertext} = \text{plaintext}^e \bmod n$$

$$D_{k-}(\text{ciphertext}) : \text{plaintext} = \text{ciphertext}^d \bmod n$$

- Example
  - **Public key (7,33), Private Key (3,33)**
  - Plaintext: 4

  - E({7,33},4) = 4^7 mod 33 = 16384 mod 33 = 16
  - D({3,33},16) = 16^3 mod 33 = 4096 mod 33 = 4

Is RSA Secure?

- $\{e,n\}$ is public information
- If you could factor n into $p*q$, then
  - could compute $\phi(n) = (p-1)(q-1)$
  - could compute $\underline{d = e^{-1} \bmod \phi(n)}$
  - would know the private key $<d,n>$!
- But: factoring large integers is hard!
  - classical problem worked on for centuries; no known reliable, fast method

Security (Cont' d)

- At present, key sizes of 1024 bits are considered to be secure, but 2048 bits is better
- Tips for making n difficult to factor
  1. p and q lengths should be similar (ex.: ~500 bits each if key is 1024 bits)
  2. both (p-1) and (q-1) should contain a "large" prime factor
  3. gcd(p-1, q-1) should be "small"
  4. d should be larger than $n^{1/4}$

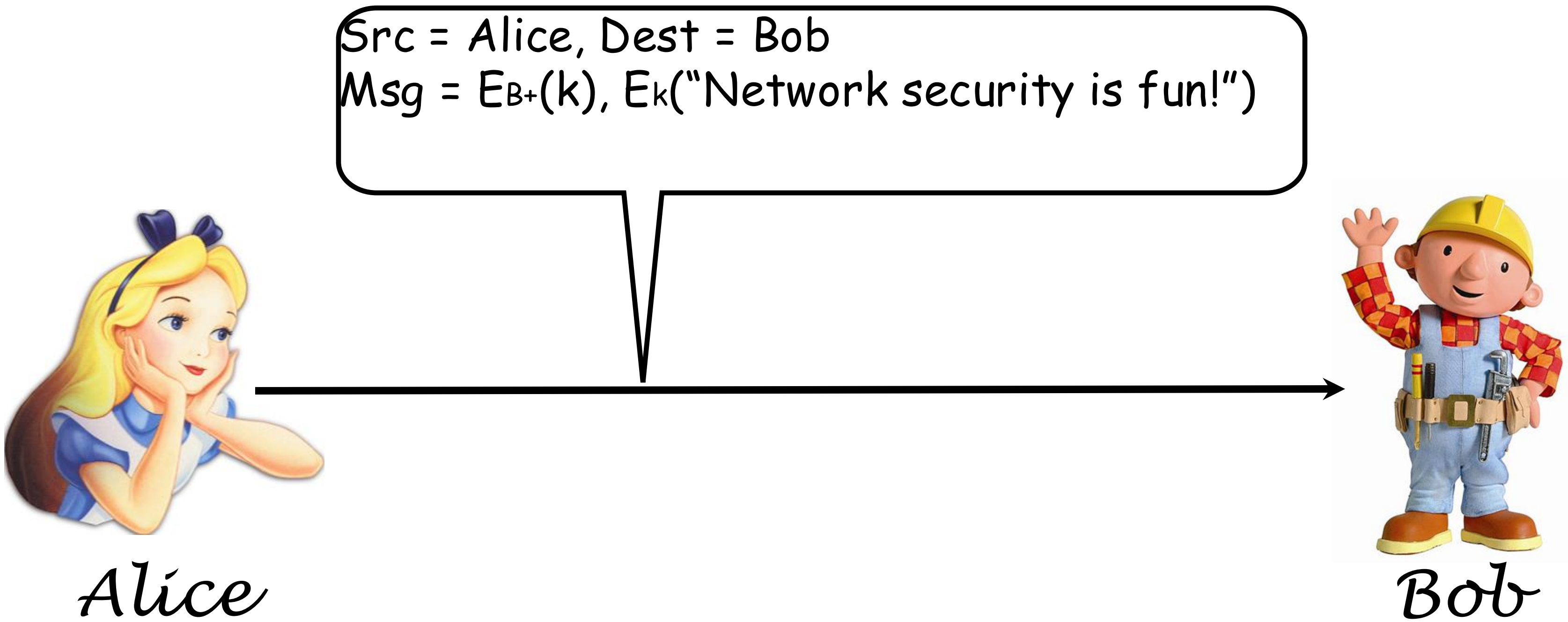
RSA

- Most public key systems use at least 1,024-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 model**

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 Cryptography

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

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

What happens if we flip the order?

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 mod 33 = 64 mod 33 = 31
  - D({7,33},31) = 31^7 mod 33 = 27,512,614,111 mod 33 = 4
- Q: *Why encrypt with private key?*
  - *Non Repudiation!*

Digital Signatures

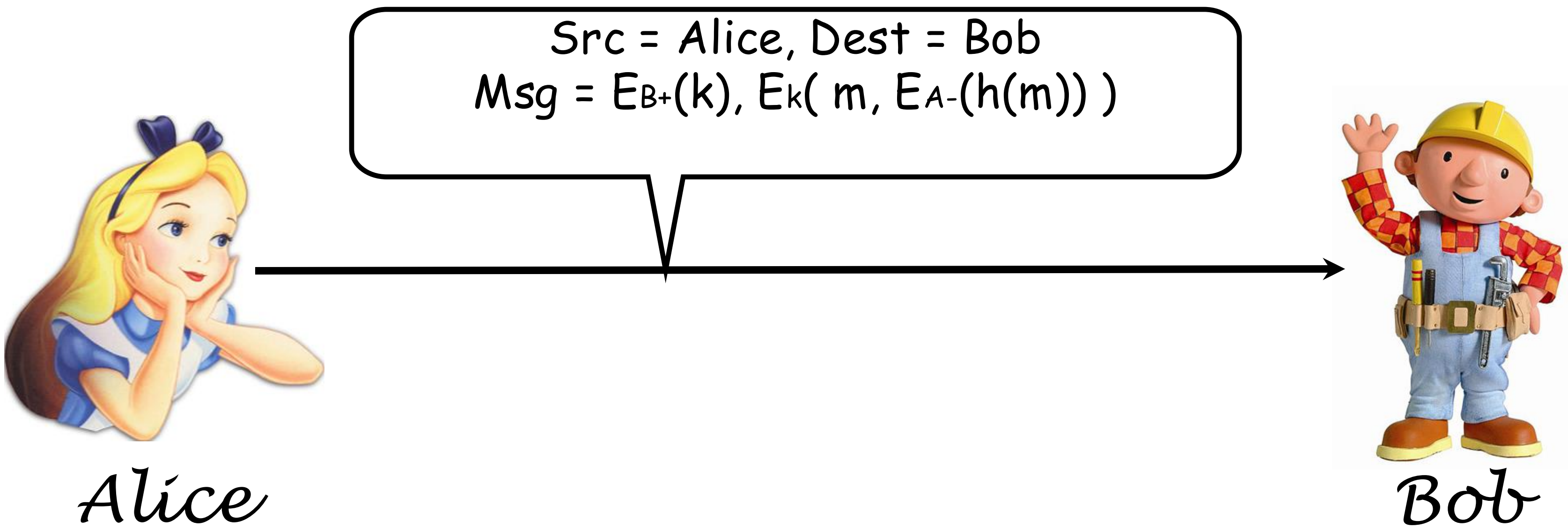
- A digital signature serves the same purpose as a real signature.
  - It is a mark that only sender can make
  - Other people can easily recognize it as belonging to the sender
- Digital signatures must be:
  - **Unforgeable**: If Alice signs message M with signature S, it is impossible for someone 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
  - Example: Code signing

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, and start moving away from SHA1

```
Dear Anthony,

{This letter is}  to introduce {you to} {Mr.}  Alfred {P.}
{ I am writing }               {to you} {-- }         {--}

Barton, the {     new        } {chief }  jewellery buyer for {our}
            {newly appointed} {senior}                       {the}

Northern {European} {  area  } . He{will take } over {the}
         { Europe } {division}     {has taken}       {-- }

responsibility for {     all      }  our interests in {watches and jewellery}
                   {the whole of }                    {jewellery and watches}

in the { area } . Please {afford} him { every  } help he {may need}
       {region}          { give }     {all the }         { needs  }

to {seek out} the most {  modern  } lines for the { top } end of the
   {  find  }          {up to date}               {high}

market. He is {empowered } to receive on our behalf { samples } of the
              {authorized}                          {specimens}

{latest} {watch and jewellery}  products, {  up   } to a { limit }
{newest} {jewellery and watch}            {subject}      {maximum}

of ten thousand dollars. He will {carry} a signed copy of this { letter }
                                 {hold }                       {document}

as proof of identity. An order with his signature, which is {appended}
                                                            {attached}

{authorizes} you to charge the cost to this company at the {   above   }
{  allows  }                                               {head office}

address. We {fully} expect that our { level} of orders will increase in
            { --  }                 {volume}

the {following} year and {trust} that the new appointment will { be  }
    {  next   }          {hope }                               {prove}

{advantageous}  to both our companies.
{an advantage}
```

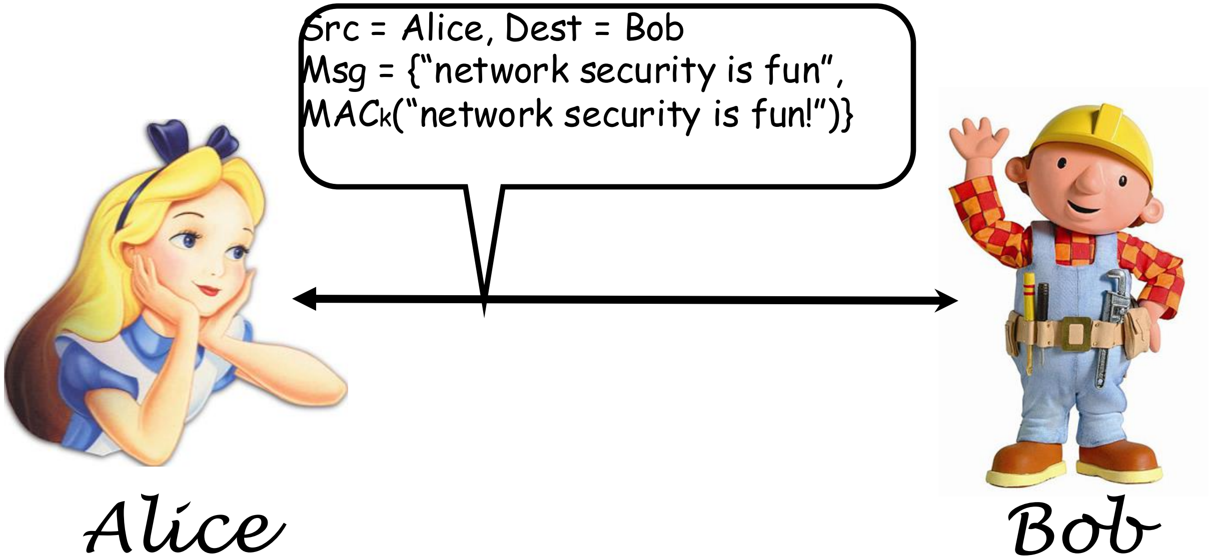
Figure 11.7 A Letter in 2^{37} Variations

(from Stallings, Crypto and Net Security)

Properties of a Digital Signature

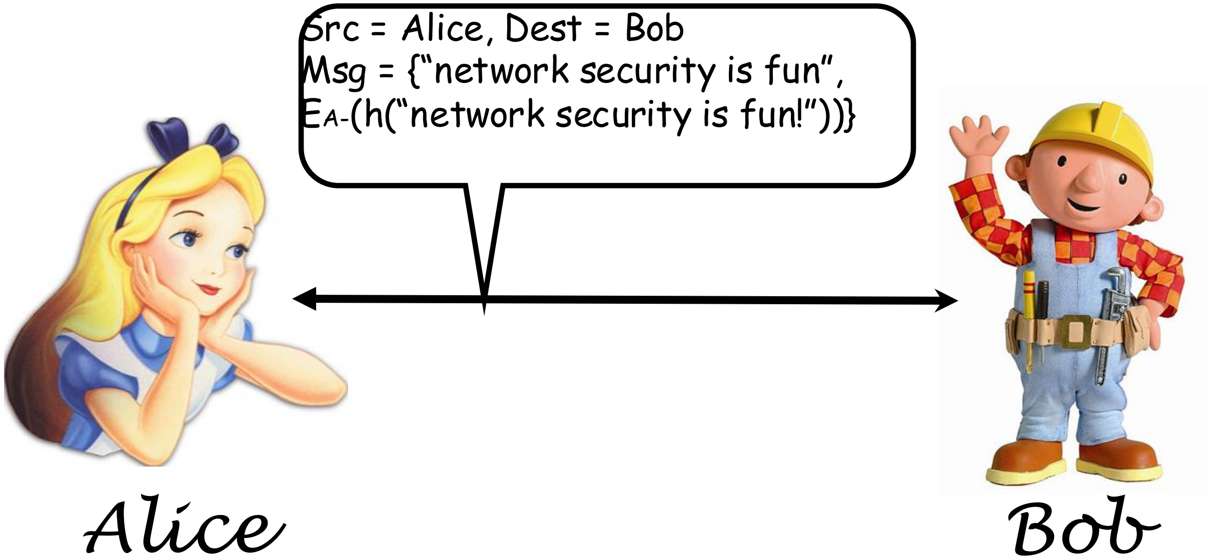
- **No forgery possible:** No one can forge a message that is purportedly from Alice
- **Authenticity check:** If you get a signed message you should be able to verify that it's really from Alice
- **No alteration/Integrity:** No party can undetectably alter a signed message
- Provides authentication,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

Non-Repudiation



MAC

Which of these offer non-repudiation?

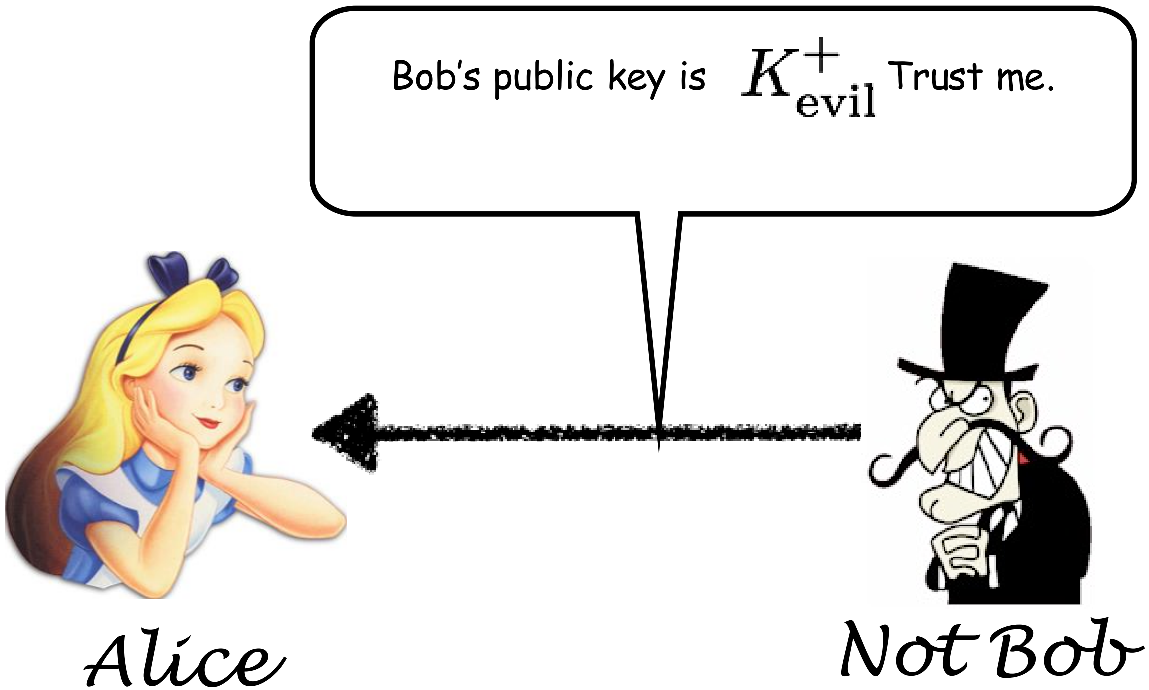


Public Key Crypto
(10,000 ft view)

- 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

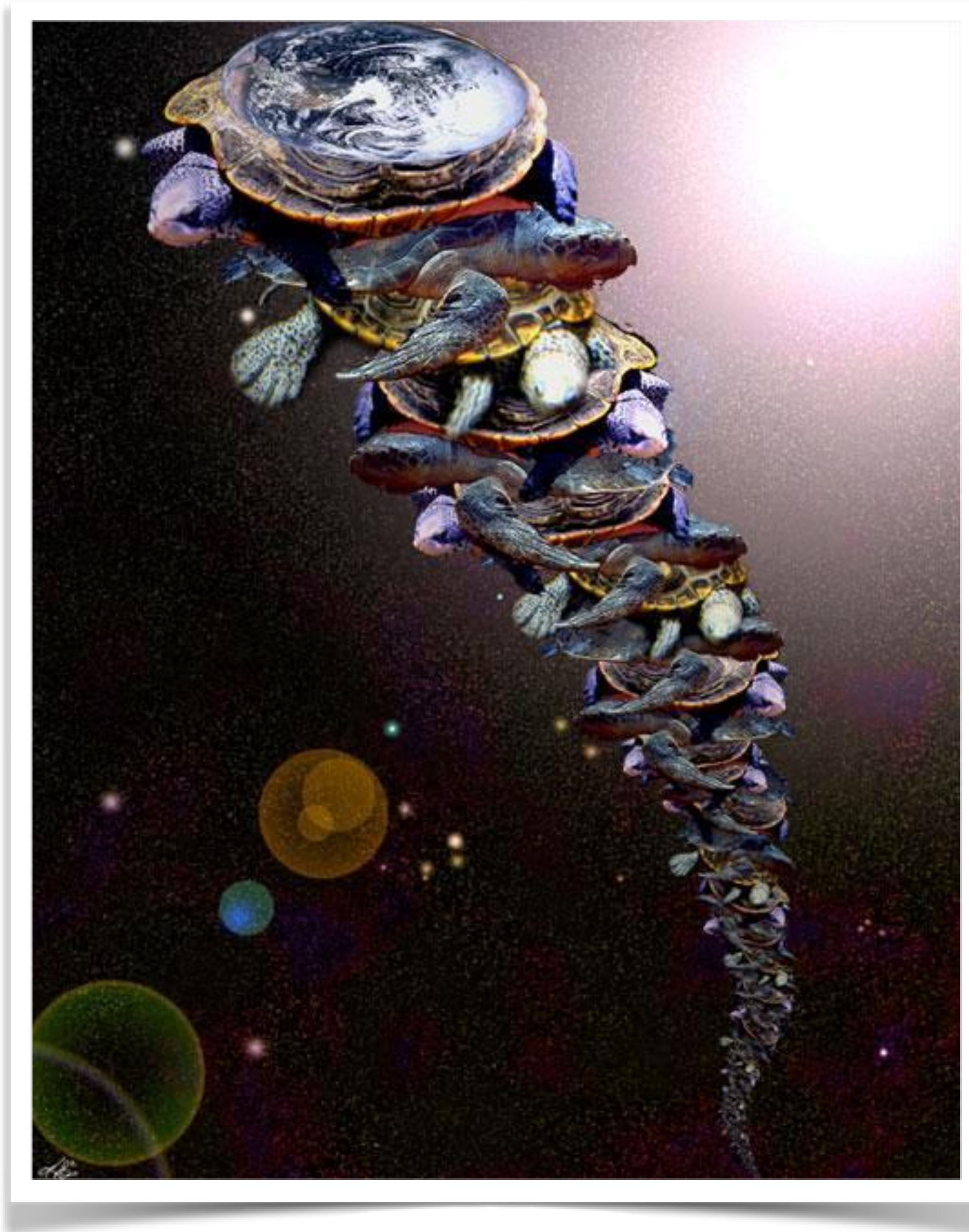
Problem? YES. *How do we know if Alice's key is really Alice's?*

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



Short answer: We can't.

It's turtles all the way down.



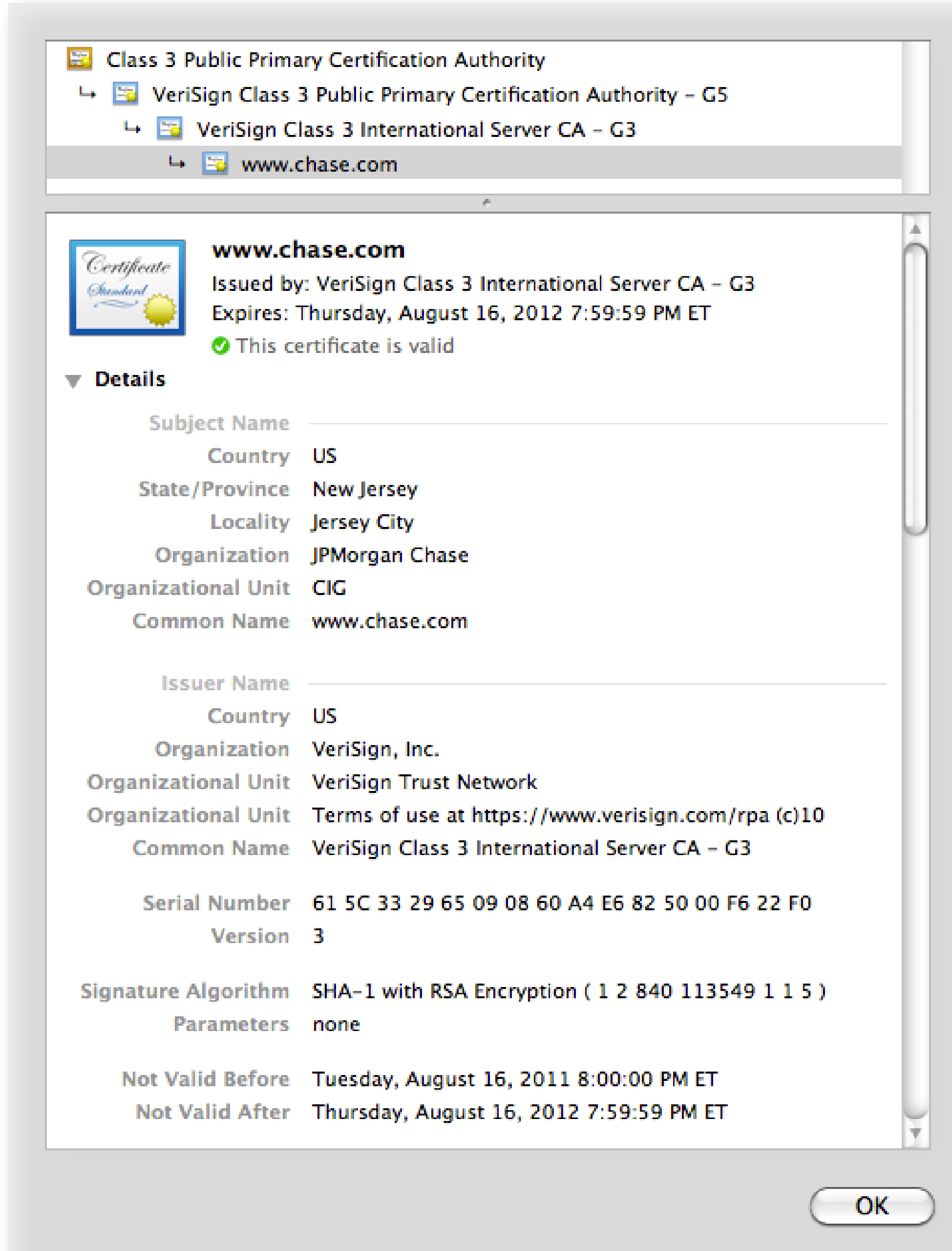
Why not just use a database?

- 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

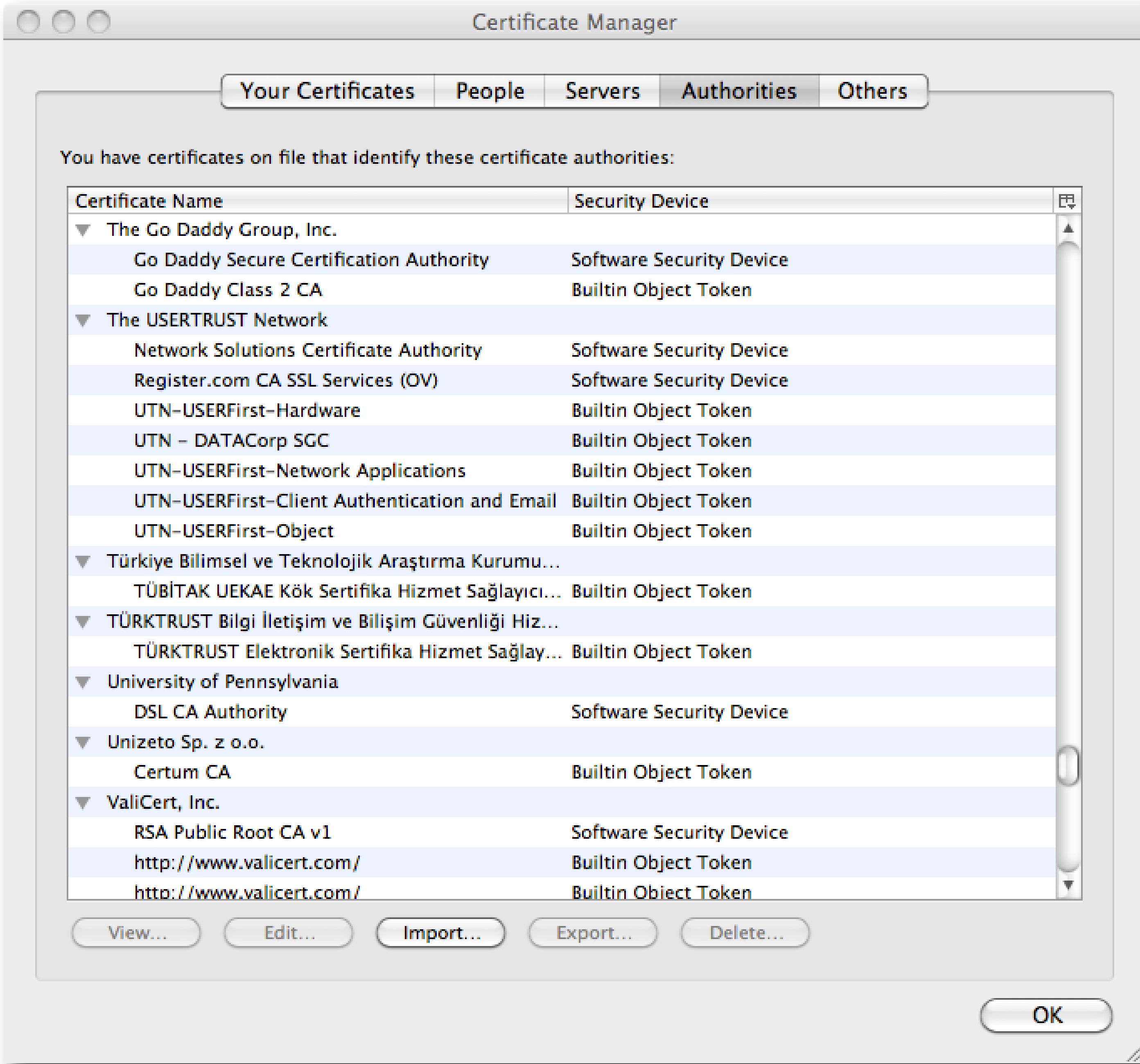


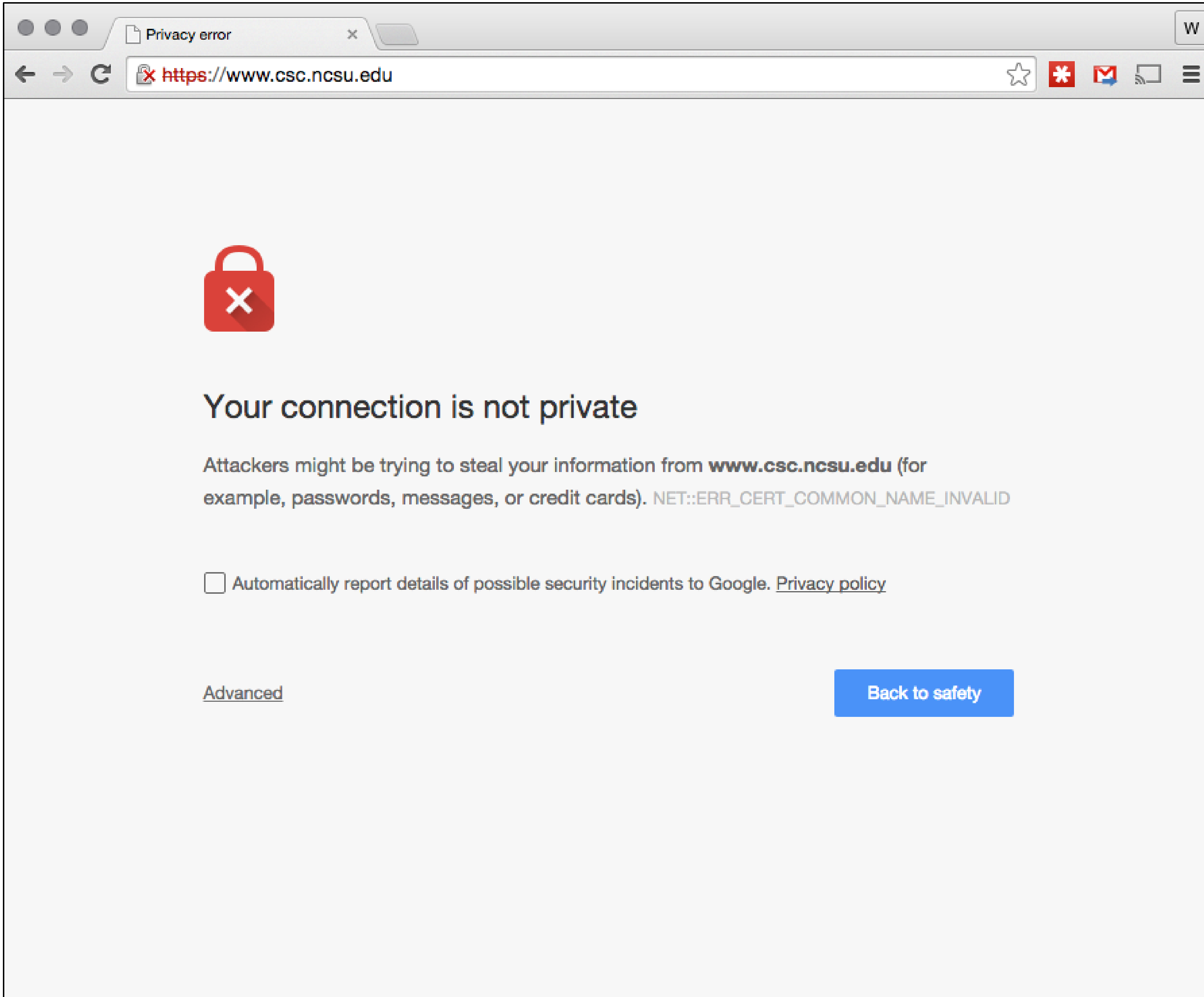
What’s a certificate?

- 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 (self-signed)*
  - … baked into your browser
  - … vetted by the browser manufacturer
  - … <u>supposedly</u> closely guarded
  - *trust anchor*
- 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

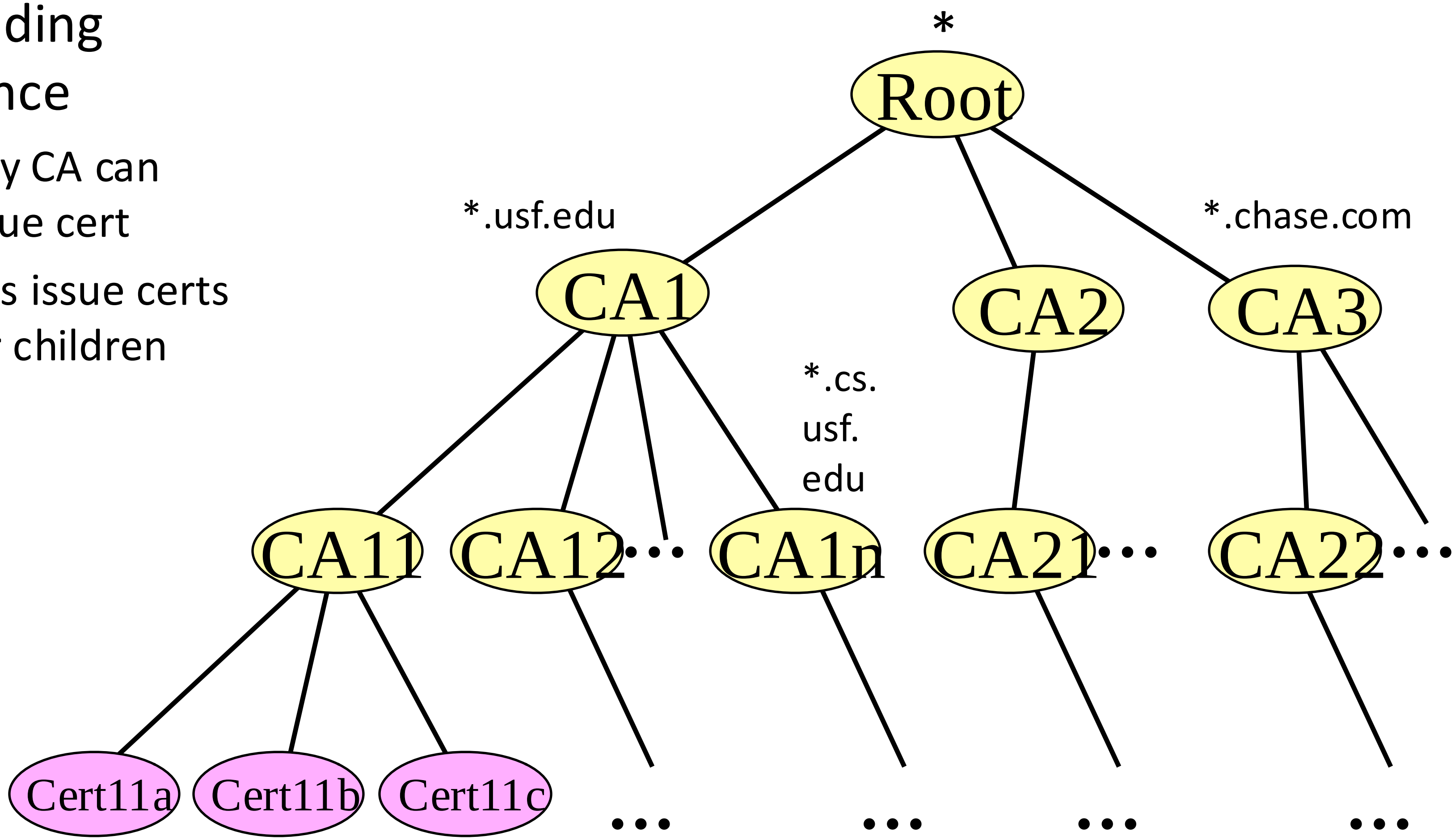

Privacy error
https://www.csc.ncsu.edu
W
Your connection is not private
Attackers might be trying to steal your information from www.csc.ncsu.edu (for example, passwords, messages, or credit cards). NET::ERR_CERT_COMMON_NAME_INVALID
Automatically report details of possible security incidents to Google.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

Public Key Infrastructure

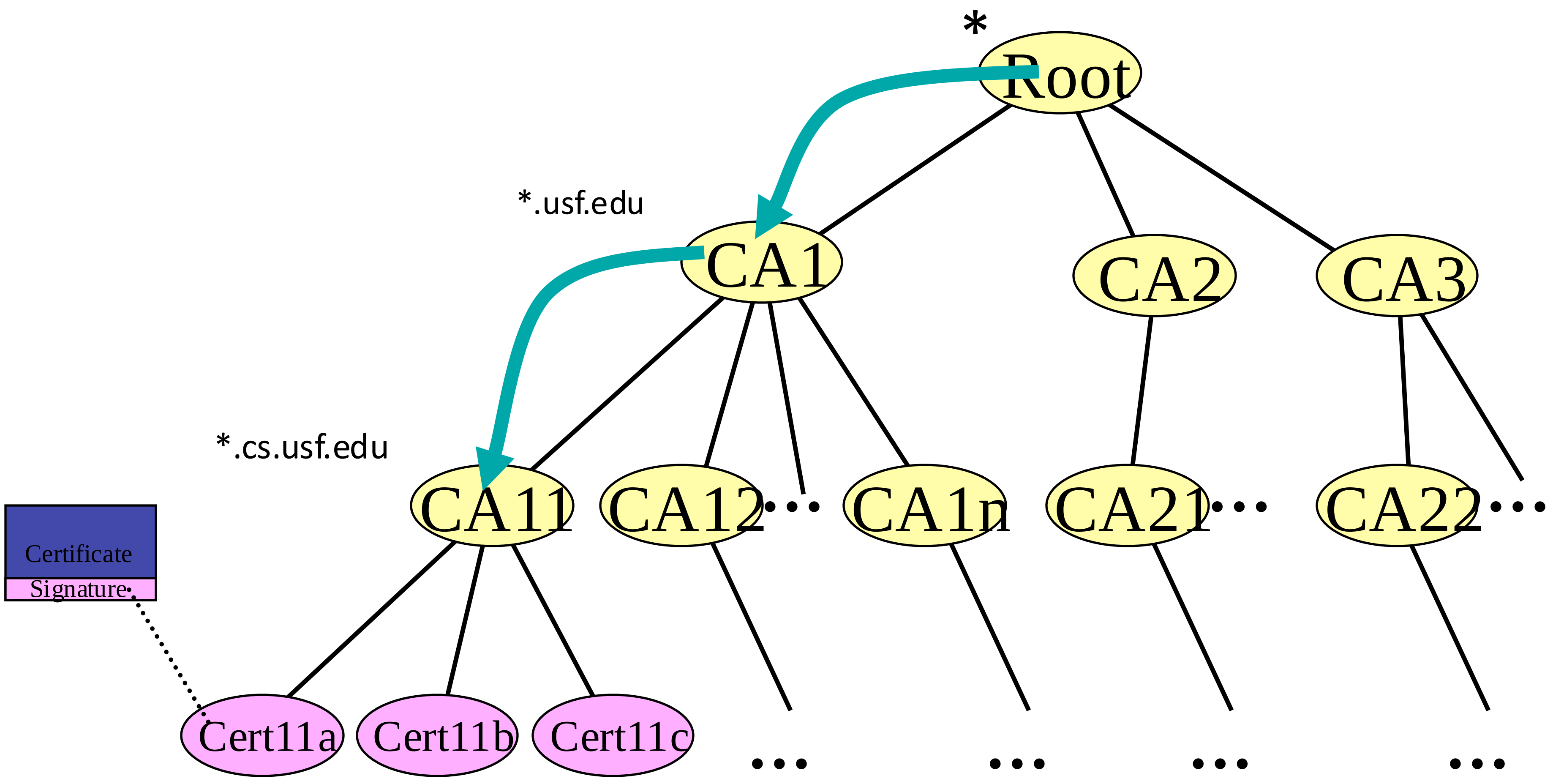
- Hierarchy of keys used to authenticate certificates
- Requires a **root of trust** (i.e., a **trust anchor**)

What is a PKI?

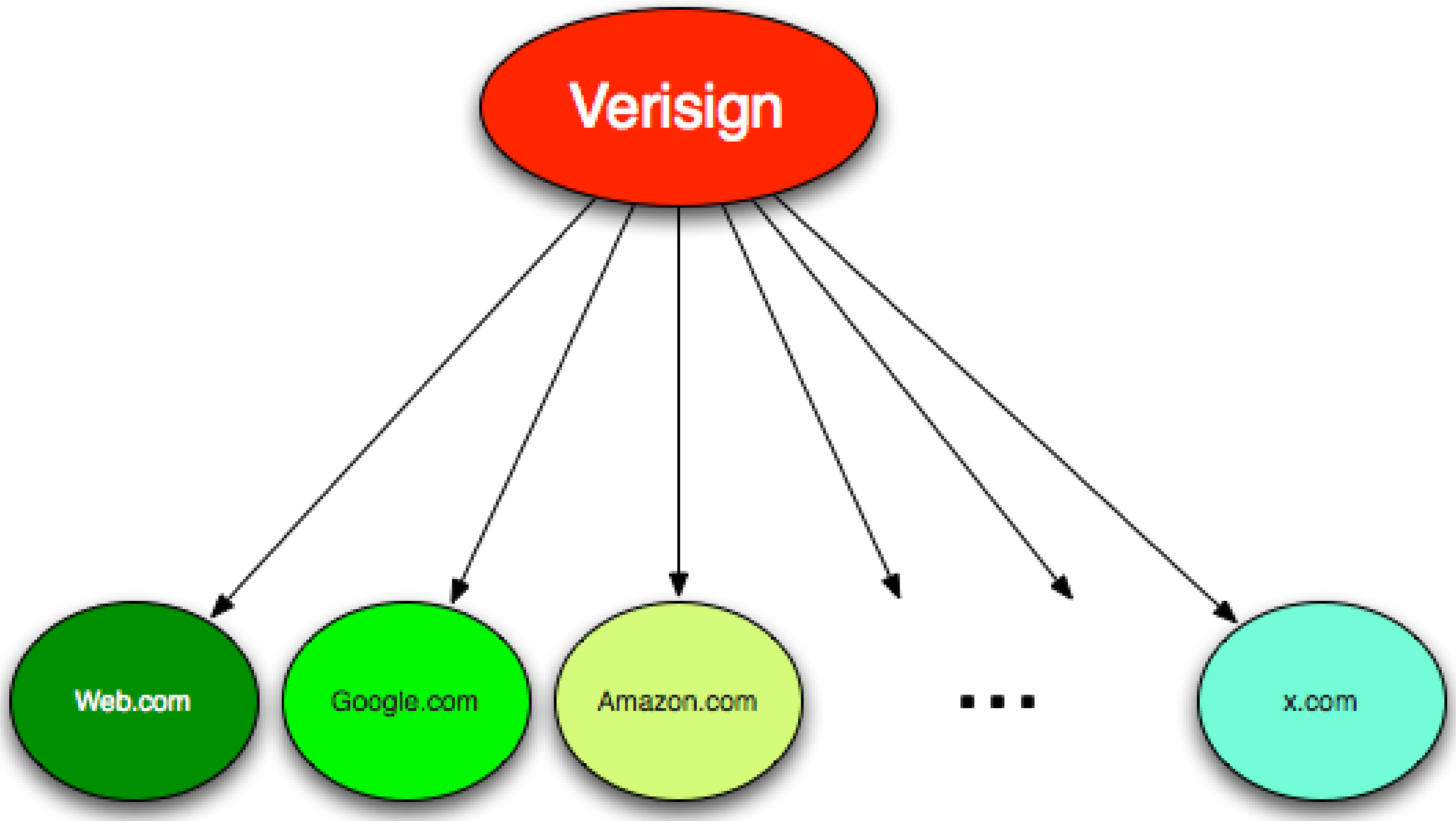
- Rooted tree of CAs
- Cascading issuance
  - Any CA can issue cert
  - CAs issue certs for children



Certificate Validation



PKIs in Reality

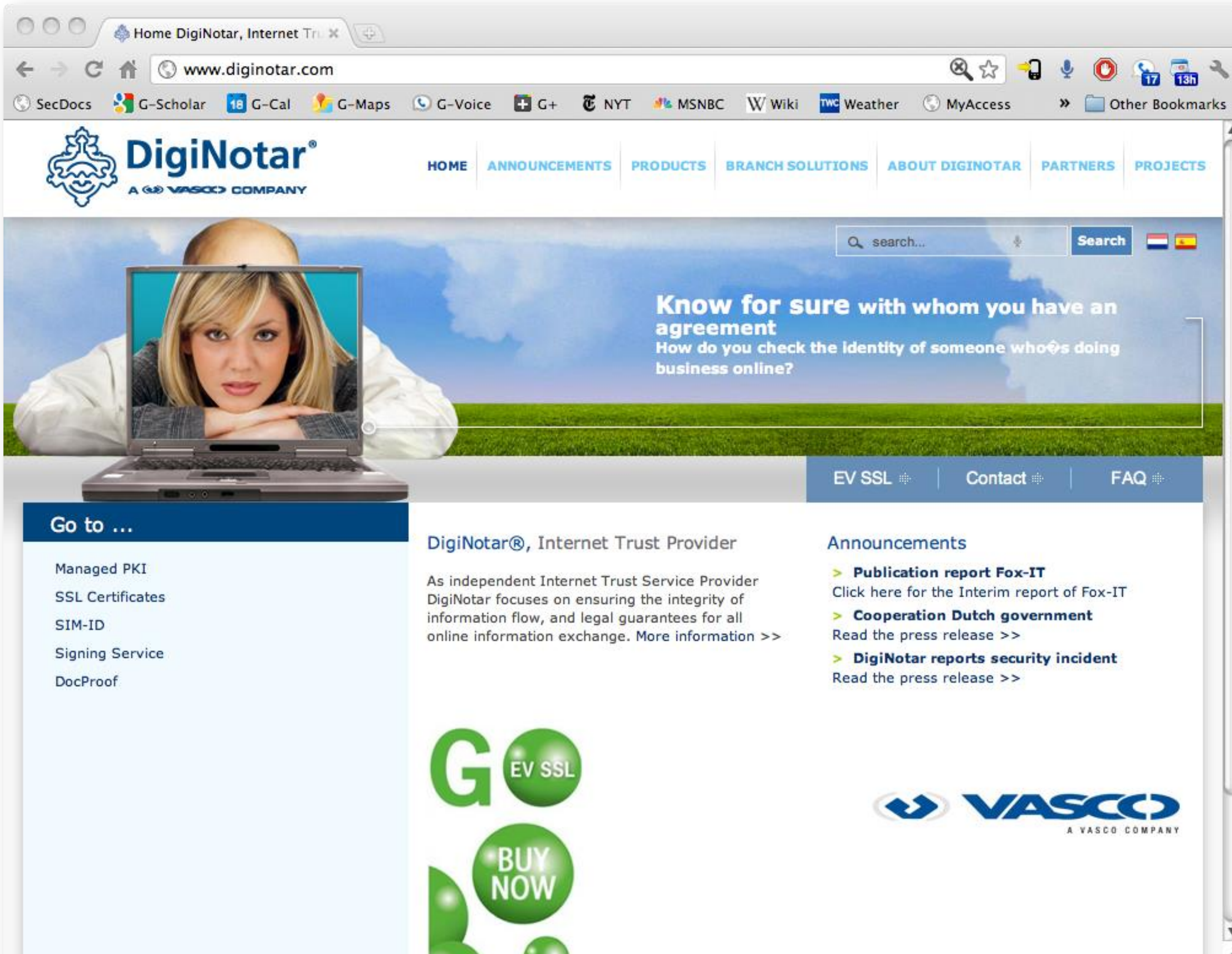


Obtaining a Certificate

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^-

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

The DigiNotar Incident

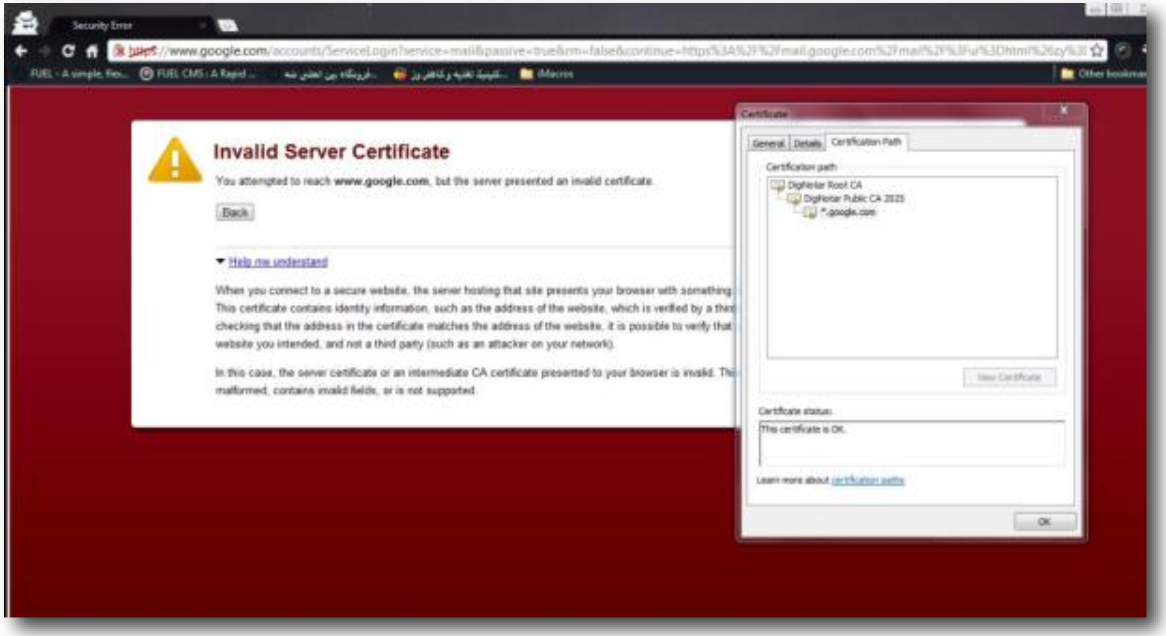


DigiNotar Incident

- 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



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 the consequences for IoT devices/apps?

The End