CSE 484: Computer Security and Privacy

Cryptography 6

Spring 2023

David Kohlbrenner

dkohlbre@cs

Thanks to Franzi Roesner, Dan Boneh, Dieter Gollmann, Dan Halperin, David Kohlbrenner, Yoshi Kohno, Ada Lerner, John Manferdelli, John Mitchell, Vitaly Shmatikov, Bennet Yee, and many others for sample slides and materials ...

Logistics

- Lab 1b coming up next week
- Homework 2 will go out today, due in 2 weeksish

Applications of Public Key Crypto

- Encryption for confidentiality
  - Anyone can encrypt a message
    - With symmetric crypto, must know secret key to encrypt
  - Only someone who knows private key can decrypt
  - Key management is simpler (or at least different)
    - Secret is stored only at one site: good for open environments
- Digital signatures for authentication
  - Can "sign" a message with your private key
- Session key establishment
  - Exchange messages to create a secret session key
  - Then switch to symmetric cryptography (why?)

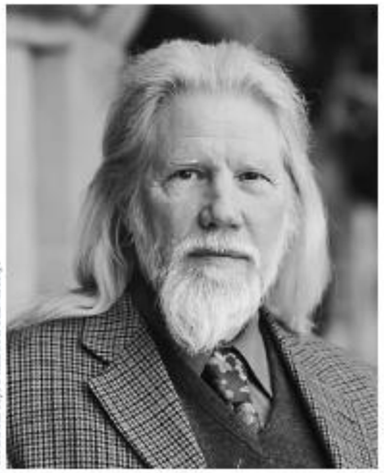
Session Key Establishment

Modular Arithmetic

- Given g and prime p, compute: $g^1 \bmod p$, $g^2 \bmod p$, … $g^{100} \bmod p$
  - For p=11, g=10
    - $10^1 \bmod 11 = 10$, $10^2 \bmod 11 = 1$, $10^3 \bmod 11 = 10$, …
    - Produces cyclic group {10, 1} (order=2)
  - For p=11, g=7
    - $7^1 \bmod 11 = 7$, $7^2 \bmod 11 = 5$, $7^3 \bmod 11 = 2$, …
    - Produces cyclic group {7,5,2,3,10,4,6,9,8,1} (order = 10)
    - g=7 is a “generator” of Z_{11}^*

Diffie-Hellman Protocol (1976)

Diffie and Hellman Receive 2015 Turing Award


Rod Searcey/Stanford University.

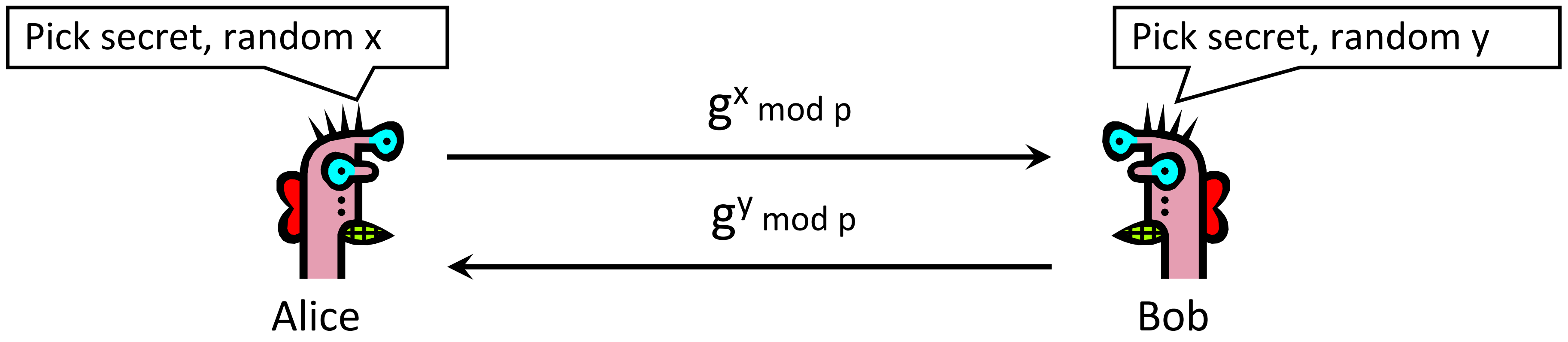
Whitfield Diffie

Linda A. Cicero/Stanford News Service.

Martin E. Hellman

Diffie-Hellman Protocol (1976)

- Alice and Bob never met and share no secrets
- Public info: p and g
  - p is a large prime, g is a **generator** of Z_p*
    - Z_p*={1, 2 … p-1}; a Z_p* i such that $a=g^i$ mod p
    - Modular arithmetic: numbers “wrap around” after they reach p



Compute $k=(g^y)^x=g^{xy}$ mod p

Compute $k=(g^x)^y=g^{xy}$ mod p

Example Diffie Hellman Computation

Why is Diffie-Hellman Secure?

- Discrete Logarithm (DL) problem:

  given $g^x \bmod p$, it's hard to extract x

  - There is no known <u>efficient</u> algorithm for doing this
  - This is <u>not</u> enough for Diffie-Hellman to be secure!

- Computational Diffie-Hellman (CDH) problem:

  given g^x and g^y, it's hard to compute $g^{xy} \bmod p$

  - … unless you know x or y, in which case it's easy

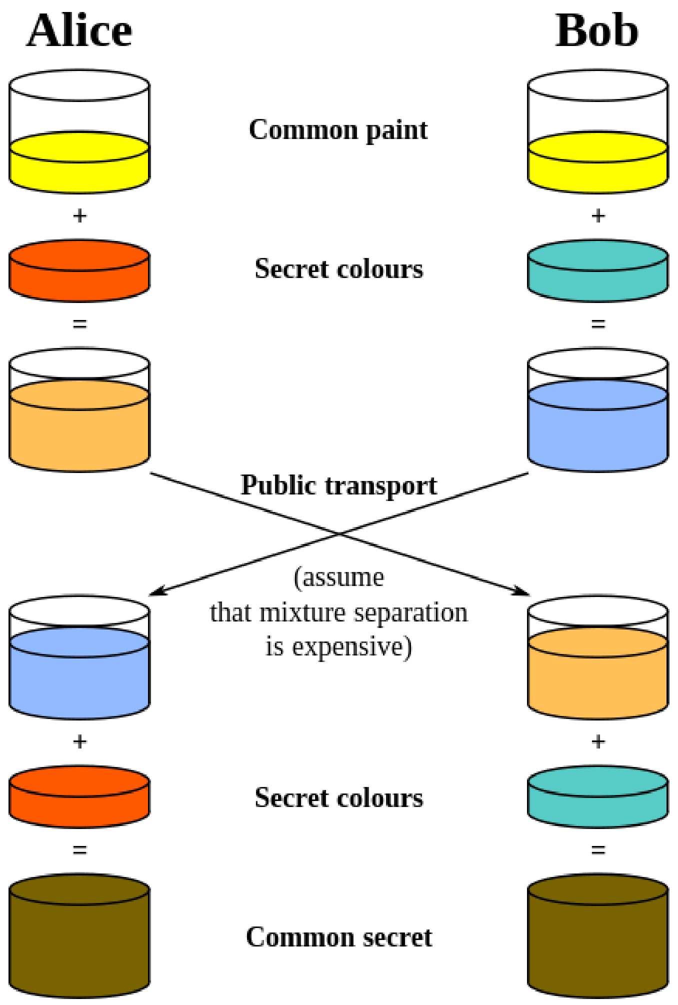
- Decisional Diffie-Hellman (DDH) problem:

  given g^x and g^y, it's hard to tell the difference between $g^{xy} \bmod p$ and $g^r \bmod p$ where r is random

More on Diffie-Hellman Key Exchange

- Important Note:
  - We have discussed discrete logs modulo integers
  - Significant advantages in using elliptic curve groups
    - Groups with some similar mathematical properties (i.e., are "groups") but have better security and performance (size) properties

Diffie-Hellman: Conceptually



Common paint: p and g

Secret colors: x and y

Send over public transport:

$g^x \bmod p$

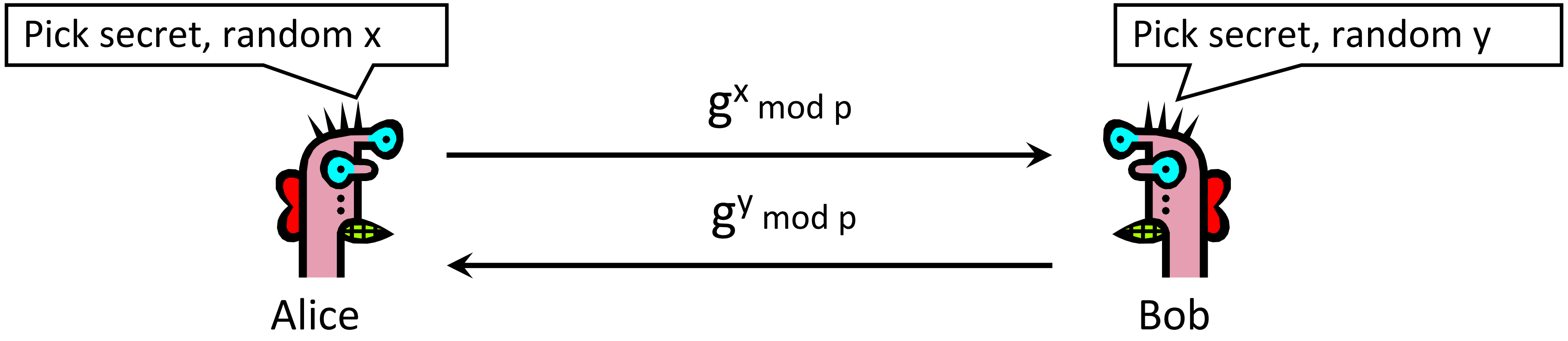
$g^y \bmod p$

Common secret: $g^{xy} \bmod p$

[from Wikipedia]

Diffie-Hellman: Canvas

- DH is a great tool, but doesn't solve every problem
- Under what circumstances (what type of adversary) is DH going to give us the full CIA(A) triad for the secret key?
- Under what circumstances might DH _not_ do that?



Compute $k=(g^y)^x=g^{xy} \bmod p$ Compute $k=(g^x)^y=g^{xy} \bmod p$

Diffie-Hellman Caveats

- Assuming DDH problem is hard (depends on choice of parameters!), Diffie-Hellman protocol is a secure key establishment protocol against passive attackers
  - Common recommendation:
    - Choose p=2q+1, where q is also a large prime
    - Choose g that generates a subgroup of order q in Z_p*
    - DDH is hard in this group
  - Eavesdropper can't tell the difference between the established key and a random value
  - In practice, often hash $g^{xy} \bmod p$, and use the hash as the key
  - Can use the new key for symmetric cryptography
- Diffie-Hellman protocol (by itself) does not provide authentication (against active attackers)
  - Person in the middle attack (also called "man in the middle attack")

Example from Earlier

- Given g and prime p, compute: $g^1 \bmod p$, $g^2 \bmod p$, … $g^{100} \bmod p$
  - For p=11, g=10
    - $10^1 \bmod 11 = 10$, $10^2 \bmod 11 = 1$, $10^3 \bmod 11 = 10$, …
    - Produces cyclic group {10, 1} (order=2)
  - For p=11, g=7
    - $7^1 \bmod 11 = 7$, $7^2 \bmod 11 = 5$, $7^3 \bmod 11 = 2$, …
    - Produces cyclic group {7,5,2,3,10,4,6,9,8,1} (order = 10)
    - g=7 is a “generator” of $Z_{11}{}^*$
  - For p=11, g=3
    - $3^1 \bmod 11 = 3$, $3^2 \bmod 11 = 9$, $3^3 \bmod 11 = 5$, …
    - Produces cyclic group {3,9,5,4,1} (order = 5) (5 is a prime)
    - g=3 generates a group of prime order

Stepping Back: Asymmetric Crypto

- We've just seen session key establishment
  - Can then use shared key for symmetric crypto
- Next: public key encryption
  - For confidentiality
- Then: digital signatures
  - For authenticity

Requirements for Public Key Encryption

- Key generation: computationally easy to generate a pair (public key PK, private key SK)
- Encryption: given plaintext M and public key PK, easy to compute ciphertext $C=E_{PK}(M)$
- Decryption: given ciphertext $C=E_{PK}(M)$ and private key SK, easy to compute plaintext M
  - Infeasible to learn anything about M from C without SK
  - Trapdoor function: Decrypt(SK,Encrypt(PK,M))=M

Some Number Theory Facts

- Euler totient function $\varphi(n)$ ($n \geq 1$) is the number of integers in the [1,n] interval that are relatively prime to n
  - Two numbers are relatively prime if their greatest common divisor (gcd) is 1
  - Easy to compute for primes: $\varphi(p) = p-1$
  - Note that $\varphi(ab) = \varphi(a)\, \varphi(b)$ if a & b are relatively prime

RSA Cryptosystem [Rivest, Shamir, Adleman 1977]

- Key generation:
  - Generate large primes p, q
    - Say, 2048 bits each (need primality testing, too)
  - Compute **n**=pq and **φ(n)**=(p-1)(q-1)
  - Choose small **e**, relatively prime to φ(n)
    - Typically, **e=3** or **$e=2^{16}+1=65537$**
  - Compute unique **d** such that ed ≡ 1 mod φ(n)
    - Modular inverse: $d \equiv e^{-1} \bmod \varphi(n)$ $\longleftarrow$ How to compute?
  - Public key = (e,n); private key = (d,n)
- Encryption of m: $c = m^e \bmod n$
- Decryption of c: $c^d \bmod n = (m^e)^d \bmod n = m$

Why is RSA Secure?

- RSA problem: given c, $n=pq$, and e such that $\gcd(e, \varphi(n))=1$, find m such that $m^e=c \bmod n$
  - In other words, recover m from ciphertext c and public key (n,e) by taking eth root of c modulo n
  - There is no known efficient algorithm for doing this *without* knowing p and q
- Factoring problem: given positive integer n, find primes $p_1, \dots, p_k$ such that $n=p_1^{e_1}p_2^{e_2}\dots p_k^{e_k}$

- If factoring is easy, then RSA problem is easy (knowing factors means you can compute d = inverse of e mod (p-1)(q-1))
  - It may be possible to break RSA without factoring n -- but if it is, we don't know how

RSA Encryption Caveats

- Encrypted message needs to be interpreted as an integer less than n
- Don’t use RSA **directly** for privacy – output is deterministic! Need to pre-process input somehow
- Plain RSA also does not provide integrity
  - Can tamper with encrypted messages

In practice, OAEP is used: instead of encrypting M, encrypt

$M \oplus G(r) \,||\, r \oplus H(M \oplus G(r))$

- r is random and fresh, G and H are hash functions

Review: RSA Cryptosystem [Rivest, Shamir, Adleman 1977]

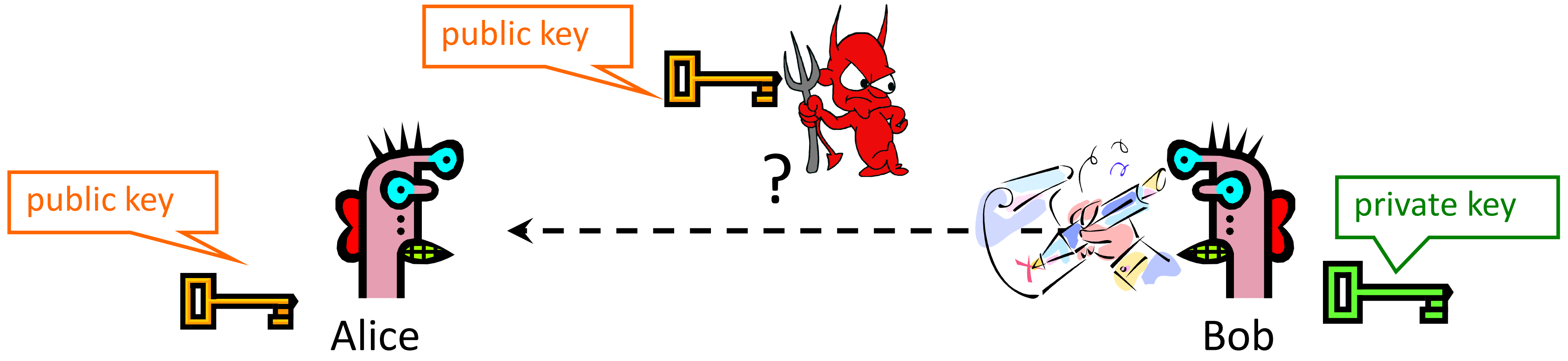
- Key generation:
  - Generate large primes p, q
    - Say, 2048 bits each (need primality testing, too)
  - Compute **n**=pq and φ**(n)**=(p-1)(q-1)
  - Choose small **e**, relatively prime to φ(n)
    - Typically, **e=3** or **e=2^{16}+1=65537**
  - Compute unique **d** such that ed ≡ 1 mod φ(n)
    - Modular inverse: d ≡ e^{-1} mod φ(n)
  - Public key = (e,n); private key = (d,n)
- Encryption of m: c = m^e mod n
- Decryption of c: c^d mod n = $(m^e)^d$ mod n = m

$\xleftarrow{\quad\quad}$ How to compute?

Actually, RSA is busted

- Math is OK, implementation isn’t
  - Yes, all the implementations
- https://blog.trailofbits.com/2019/07/08/fuck-rsa/
- Sorry I just spent time teaching it to you
  - Maybe you would’ve preferred projected coordinate math on elliptic curves?

Digital Signatures: Basic Idea



Given: Everybody knows Bob's public key
Only Bob knows the corresponding private key

Goal: Bob sends a "digitally signed" message

1. To compute a signature, must know the private key
2. To verify a signature, only the public key is needed

RSA Signatures

- Public key is **(n,e)**, private key is **(n,d)**
- To sign message m: s = m^d mod n
  - Signing & decryption are same **underlying** operation in RSA
  - It's infeasible to compute **s** on **m** if you don't know **d**
- To verify signature s on message m:

  verify that s^e mod n = $(m^d)^e$ mod n = m
  - Just like encryption (for RSA primitive)
  - Anyone who knows **n** and **e** (public key) can verify signatures produced with d (private key)
- In practice, also need padding & hashing
  - Without padding and hashing: Consider multiplying two signatures together
  - Standard padding/hashing schemes exist for RSA signatures

DSS Signatures

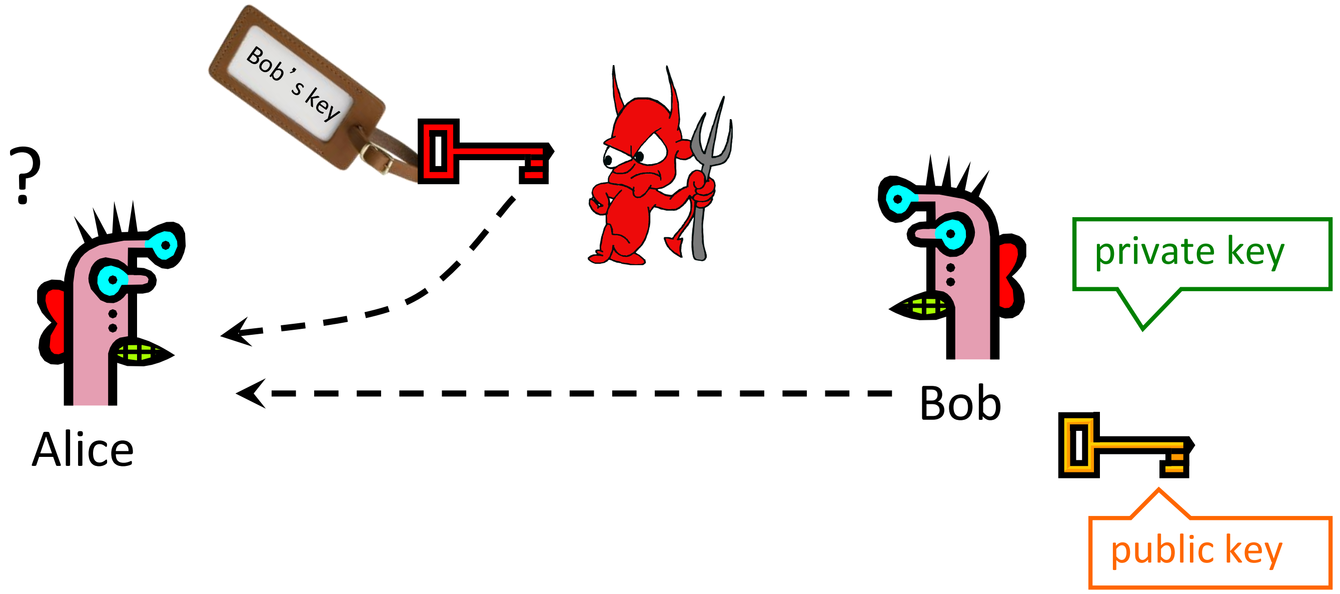
- Digital Signature Standard (DSS)
  - U.S. government standard (1991, most recent rev. 2013)
- Public key: (p, q, g, $y=g^x$ mod p), private key: x
- Each signing operation picks a new random value, to use during signing. Security breaks if two messages are signed with that same value.
- Security of DSS requires hardness of discrete log
  - If could solve discrete logarithm problem, would extract x (private key) from g^x mod p (public key)
- Again: We've discussed discrete logs modulo integers; significant advantages to using elliptic curve groups instead.

Post-Quantum

- If quantum computer become a reality
  - It becomes much more efficient to break conventional asymmetric encryption schemes (e.g., factoring becomes “easy”)

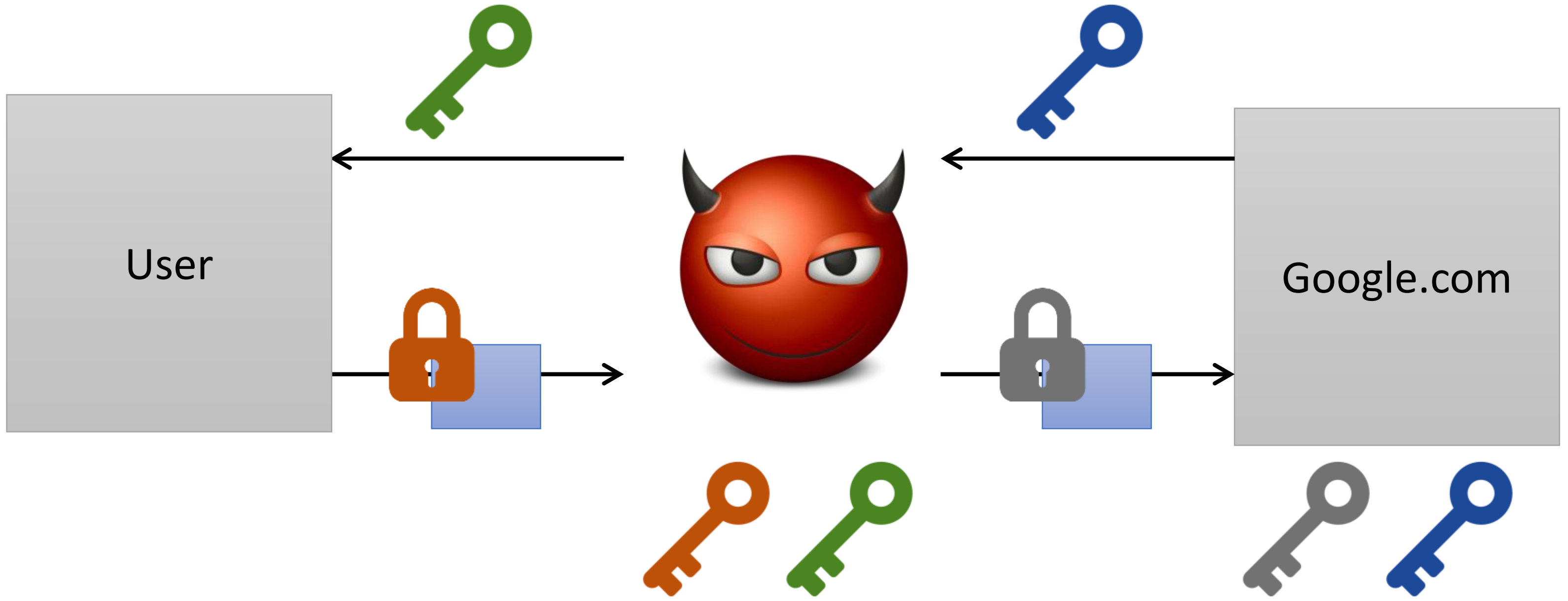
- There exists efforts to make quantum-resilient asymmetric encryption schemes
  - (Check out NIST’s PQC competition!)

Authenticity of Public Keys



Problem: How does Alice know that the public key they received is really Bob's public key?

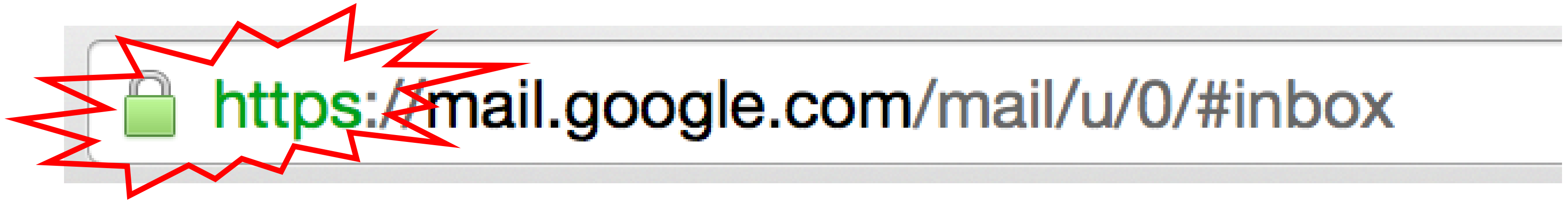
Threat: Person-in-the Middle



Distribution of Public Keys

- Public announcement or public directory
  - Risks: forgery and tampering
- Public-key certificate
  - Signed statement specifying the key and identity
    - sig_{CA}(“Bob”, PK_B)
    - Additional information often signed as well (e.g., expiration date)
- Common approach: certificate authority (CA)
  - Single agency responsible for certifying public keys
  - After generating a private/public key pair, user proves their identity and knowledge of the private key to obtain CA’s certificate for the public key (offline)
  - Every computer is pre-configured with CA’s public key

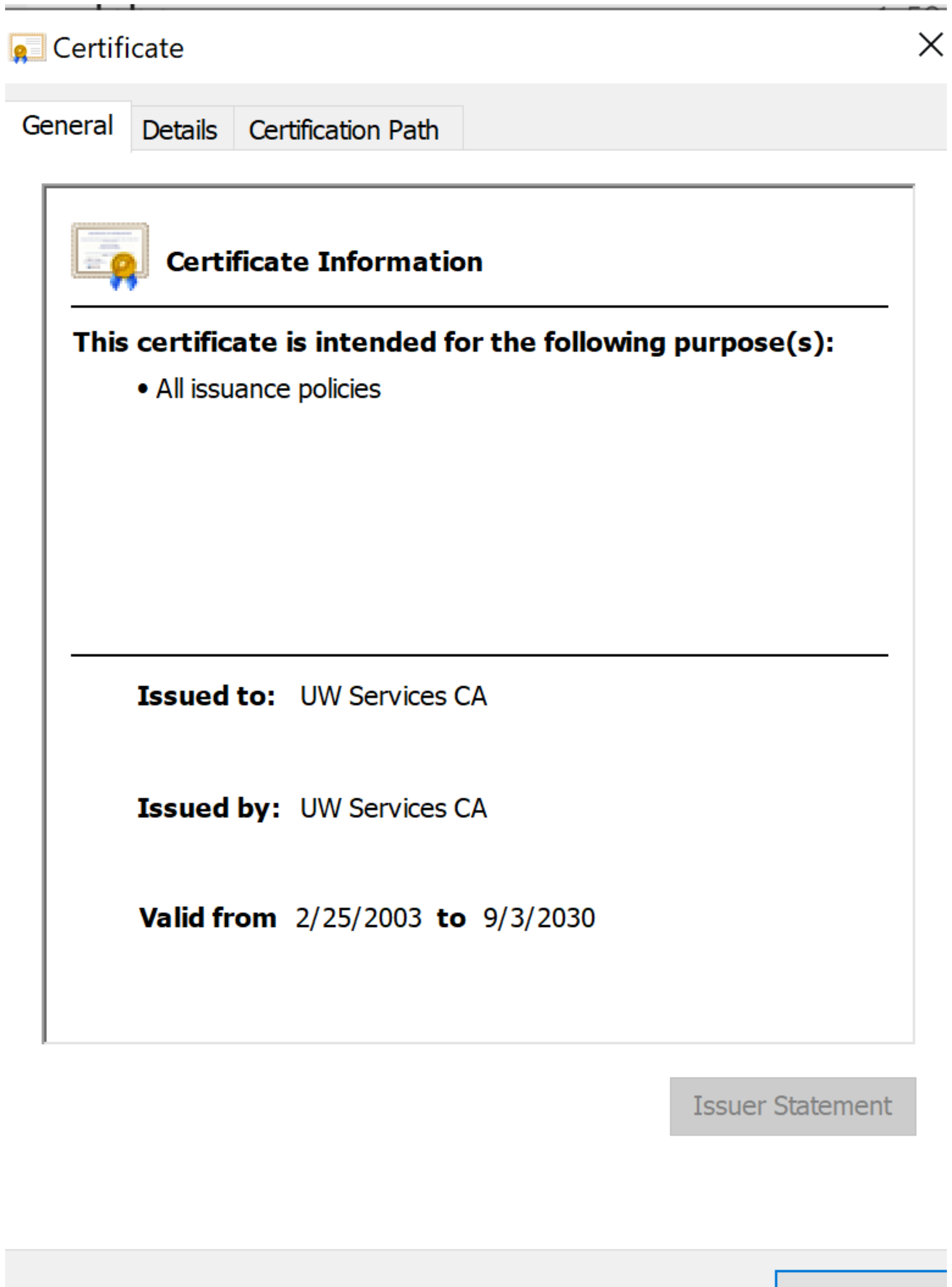
You encounter this every day...


https://mail.google.com/mail/u/0/#inbox

SSL/TLS: Encryption & authentication for connections

SSL/TLS High Level

- SSL/TLS consists of two protocols
  - Familiar pattern for key exchange protocols
- Handshake protocol
  - Use public-key cryptography to establish a shared secret key between the client and the server
- Record protocol
  - Use the secret symmetric key established in the handshake protocol to protect communication between the client and the server



Certificate

General | Details | Certification Path

Certificate Information

This certificate is intended for the following purpose(s):

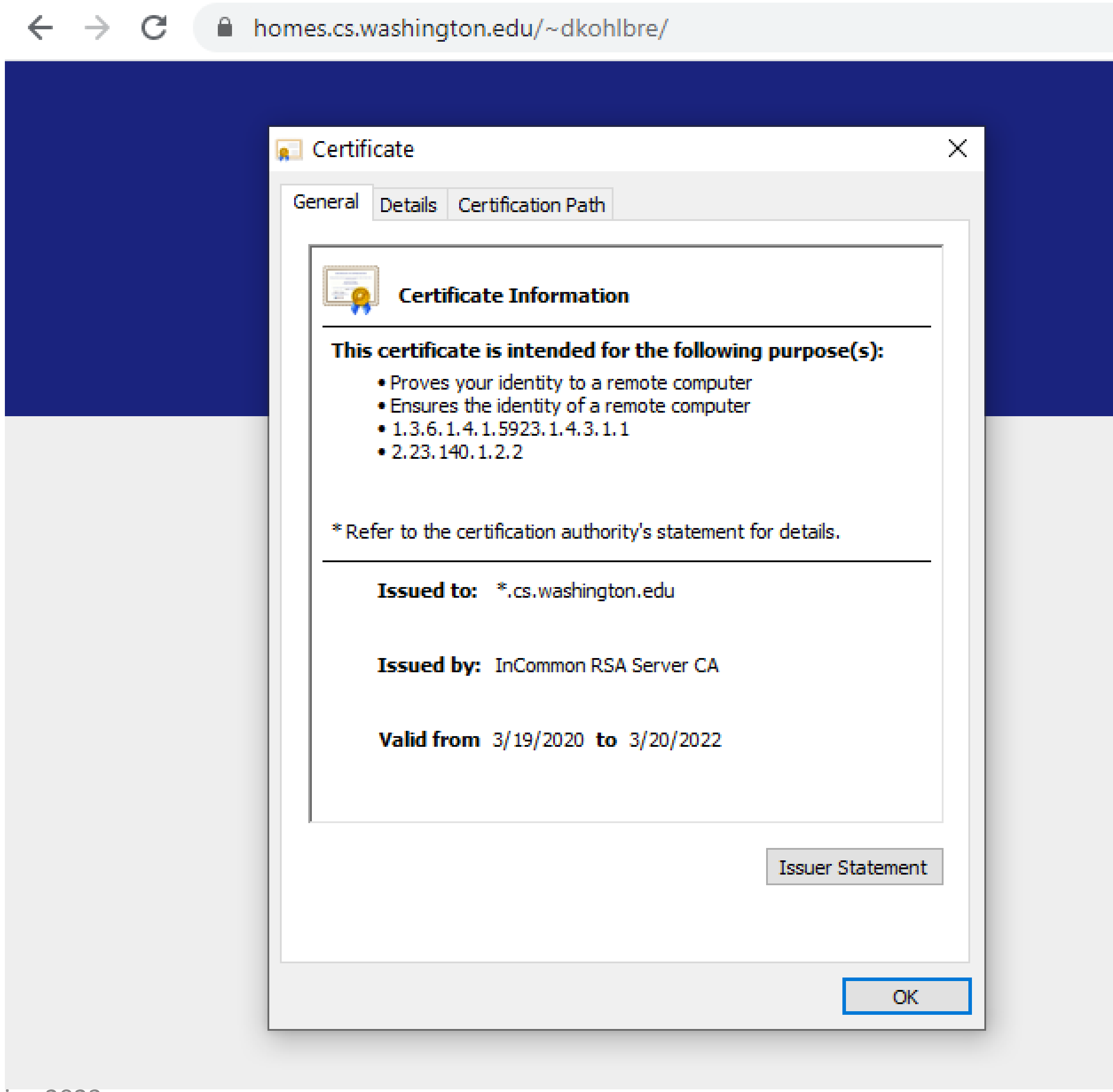
- All issuance policies

Issued to: UW Services CA

Issued by: UW Services CA

Valid from 2/25/2003 **to** 9/3/2030

Issuer Statement



homes.cs.washington.edu/~dkohlbre/

Certificate

General | Details | Certification Path

Certificate Information

This certificate is intended for the following purpose(s):

- Proves your identity to a remote computer
- Ensures the identity of a remote computer
- 1.3.6.1.4.1.5923.1.4.3.1.1
- 2.23.140.1.2.2

* Refer to the certification authority's statement for details.

Issued to: *.cs.washington.edu

Issued by: InCommon RSA Server CA

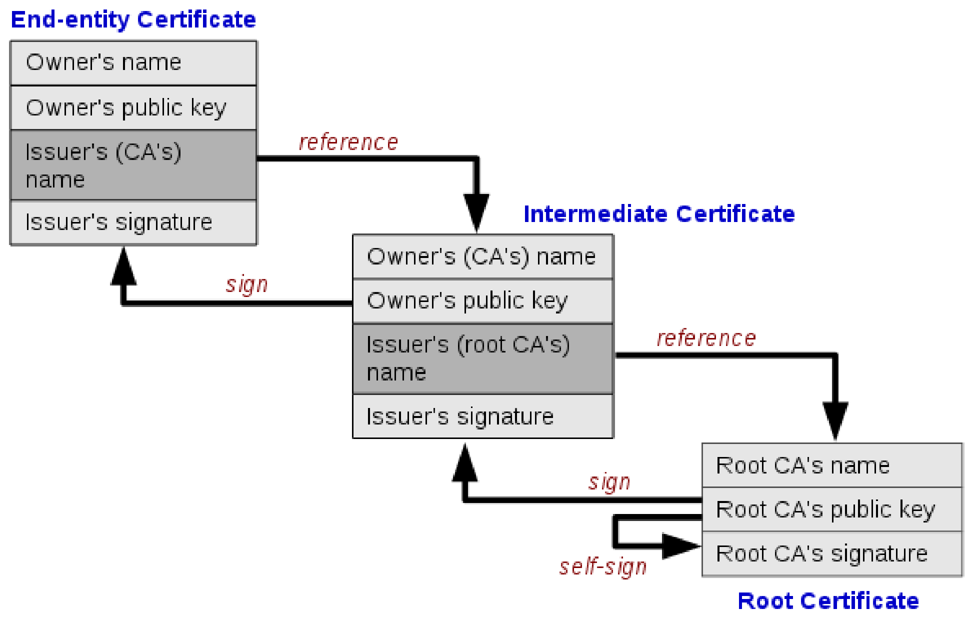
Valid from 3/19/2020 **to** 3/20/2022

Issuer Statement

OK

Hierarchical Approach

- Single CA certifying every public key is impractical
- Instead, use a trusted root authority (e.g., Verisign)
  - Everybody must know the root's public key
  - Instead of single cert, use a certificate chain
    - $\text{sig}_{\text{Verisign}}$(“AnotherCA”, $\text{PK}_{\text{AnotherCA}}$), $\text{sig}_{\text{AnotherCA}}$(“Alice”, PK_{A})
  - Not shown in figure but important:
    - Signed as part of each cert is whether party is a CA or not



  - What happens if root authority is ever compromised?