# CSE 484 / CSE M 584: Hash Functions + Asymmetric Cryptography

Winter 2024

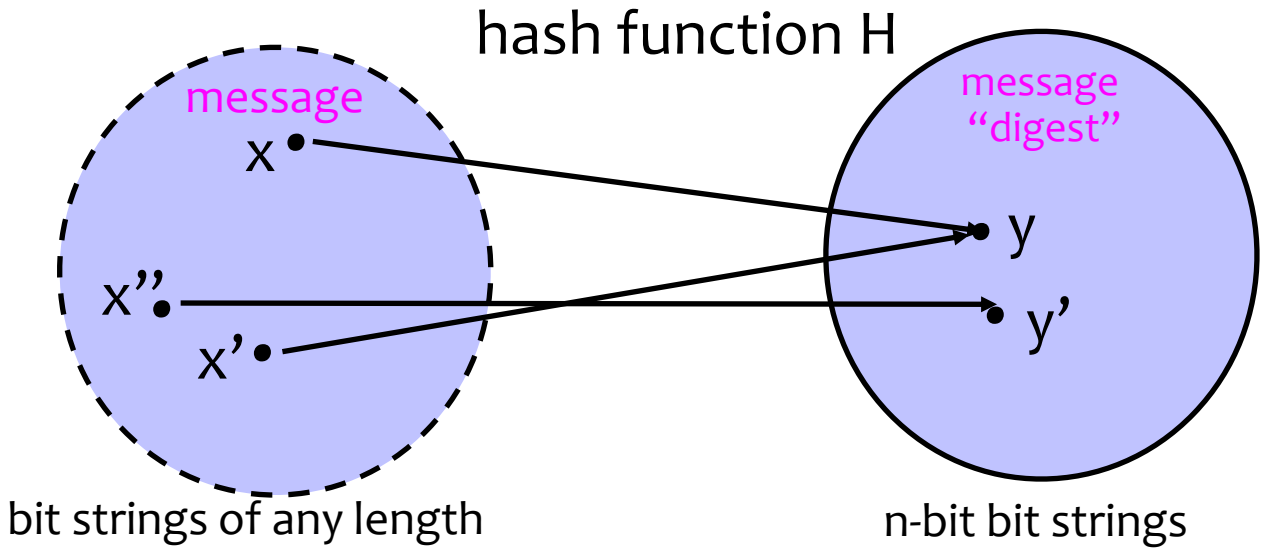
Tadayoshi (Yoshi) Kohno
yoshi@cs

UW Instruction Team: David Kohlbrenner, Yoshi Kohno, Franziska Roesner. Thanks to Dan Boneh, Dieter Gollmann, Dan Halperin, John Manferdelli, John Mitchell, Vitaly Shmatikov, Bennet Yee, and many others for sample slides and materials …

# Announcements

- HW2: Due tomorrow (extended)
- Wednesday, Feb 7 (Zoom)
  - Guest Lecture: Gennie Gebhart (EFF + UW)
- Friday, Feb 23 (in person)
  - Guest Lecture: Jim O'Leary (Signal)

# (Review) Hash Functions: Main Idea




- Hash function H is a lossy compression function
  - Collision: h(x)=h(x') for distinct inputs x, x'
- H(x) should look "random"
  - Every bit (almost) equally likely to be 0 or 1
- Cryptographic hash function needs a few properties…

# Hashing vs. Encryption

- Hashing is one-way. There is no "un-hashing"
  - A ciphertext can be decrypted with a decryption key… hashes have no equivalent of "decryption"
- Hash(x) looks "random" but can be compared for equality with Hash(x')
  - Hash the same input twice → same hash value
  - Encrypt the same input twice → different ciphertexts
- Crytographic hashes are also known as "cryptographic checksums" or "message digests"

# Application: Password Hashing

- Instead of user password, store hash(password)
- When user enters a password, compute its hash and compare with the entry in the password file
- Why is hashing better than encryption here?

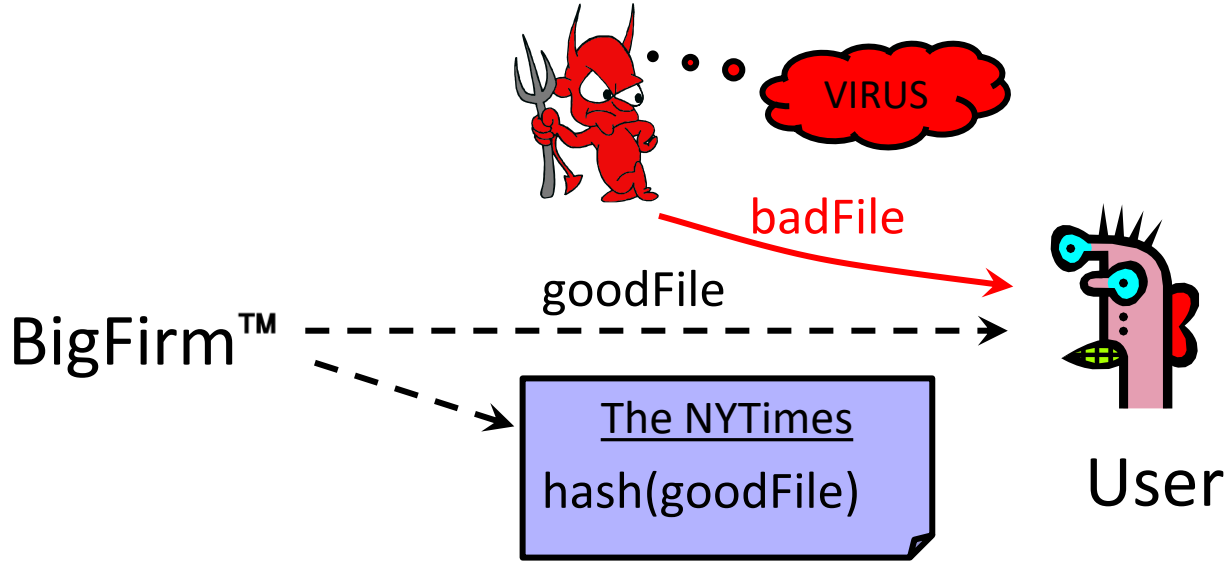
- System does not store actual passwords
- Don't need to worry about where to store the key
- Cannot go from hash to password

# Application: Password Hashing

- Which property do we need?
  - One-wayness?
  - (At least weak) Collision resistance?
  - Both?

- This is not the whole story on password storage; we'll return to this later in the course.

# Application: Software Integrity




Goal: Software manufacturer wants to ensure file is received by users without modification.

Idea: given goodFile and hash(goodFile), very hard to find badFile such that hash(goodFile)=hash(badFile)

# Application: Software Integrity

- Which property do we need?
  - One-wayness?
  - (At least weak) Collision resistance?
  - Both?

# Which Property Do We Need?

## One-wayness, Collision Resistance, Weak CR?

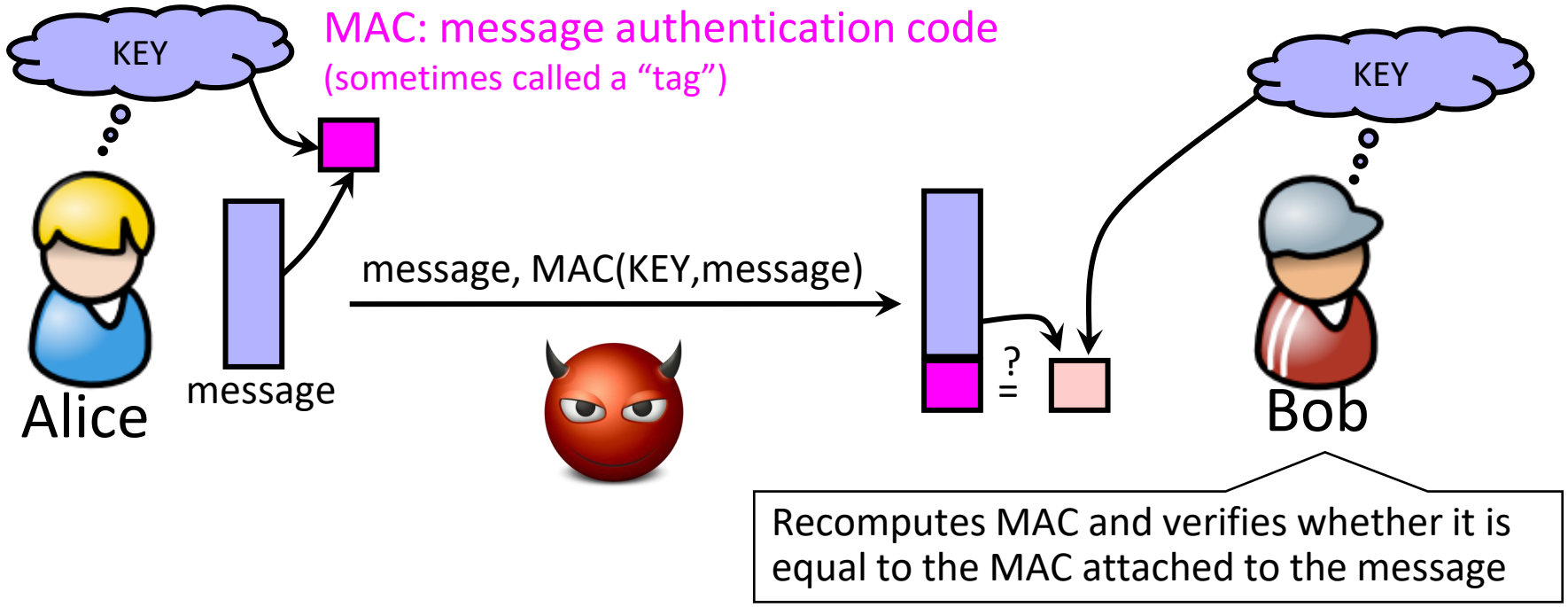
- UNIX passwords stored as hash(password)
  - **One-wayness:** hard to recover the/a valid password
- Integrity of software distribution
  - **Weak collision resistance**
  - But software images are not really random… may need **full collision resistance** if considering malicious developers

# Common Hash Functions

- **SHA-2: SHA-256, SHA-512, SHA-224, SHA-384**
- **SHA-3: standard released by NIST in August 2015**
- MD5 – Don't use for security!
  - 128-bit output
  - Designed by Ron Rivest, used very widely
  - Collision-resistance broken (summer of 2004)
- SHA-1 (Secure Hash Algorithm) – Don't use for security!
  - 160-bit output
  - US government (NIST) standard as of 1993-95
  - Theoretically broken 2005; practical attack 2017!

# Recall: Achieving Integrity

Message authentication schemes: A tool for protecting integrity.




Integrity and authentication: only someone who knows KEY can compute correct MAC for a given message.

# MAC with SHA3

- SHA3(Key || Message)
- Nice and simple ☺
- Previous hash functions couldn't quite be used in this way (see: length extension attack)
  - HMAC construction (FYI)

- Why not encryption? (Historical reasons)
  - Hashing is faster than block ciphers in software
  - Can easily replace one hash function with another
  - There used to be US export restrictions on encryption

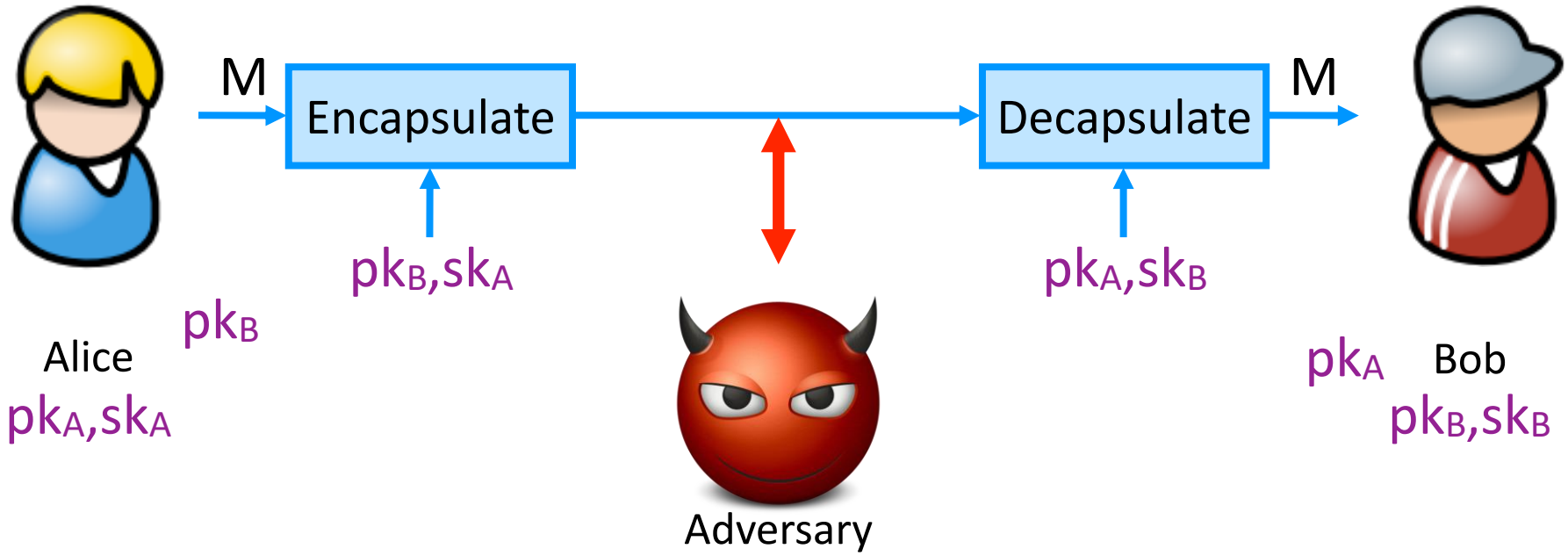
# Flavors of Cryptography

- Symmetric cryptography
  - Both communicating parties have access to a shared random string K, called the key.

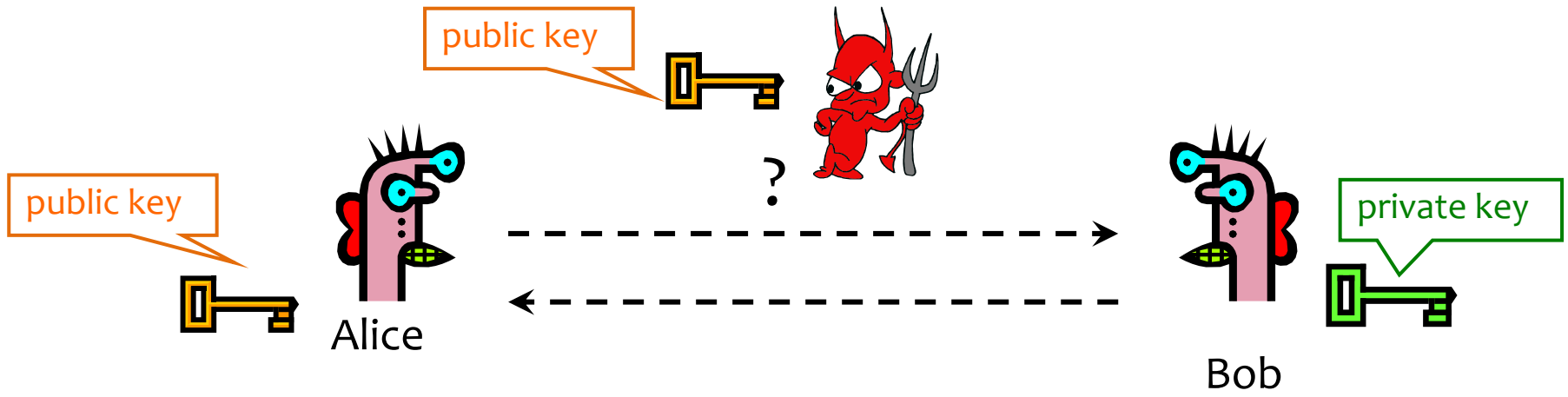
- Asymmetric cryptography
  - Each party creates a public key pk and a secret key sk.

# Asymmetric Setting

Each party creates a public key pk and a secret key sk.



# Public Key Crypto: Basic Problem




Ignore for now: How do we know it's REALLY Bob's??

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

Goals: 1. Alice wants to send a secret message to Bob
2. Bob wants to authenticate themself

# 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?)

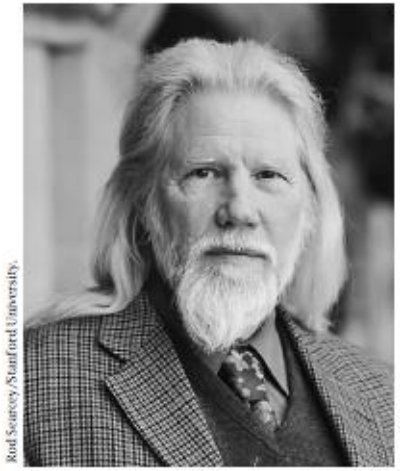
# Session Key Establishment

# 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)
      - Numbers "wrap around" after they reach p
    - g=7 is a "generator" of $Z_{11}^*$

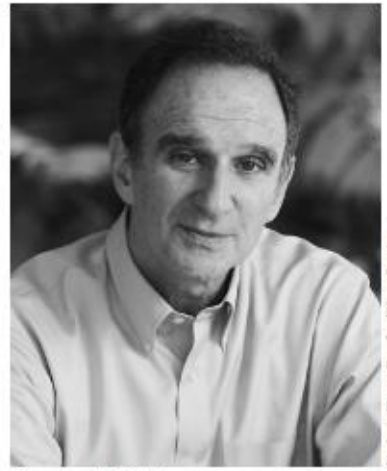
# Diffie-Hellman Protocol (1976)

## Diffie and Hellman Receive 2015 Turing Award



Rod Searcey/Stanford University.

Whitfield Diffie

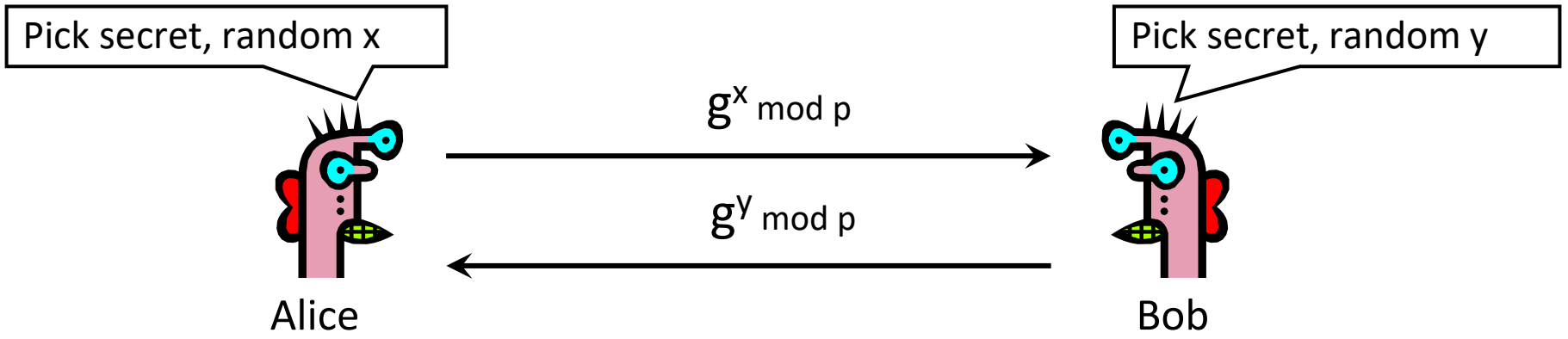


Linda A. Cicero/Stanford News Service.

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 $Z_p^*$
    - $Z_p^*=\{1, 2 \ldots p-1\}$; a is in $Z_p^*$ if there is an i such that $a=g^i \bmod p$




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

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

# Example Diffie Hellman Computation

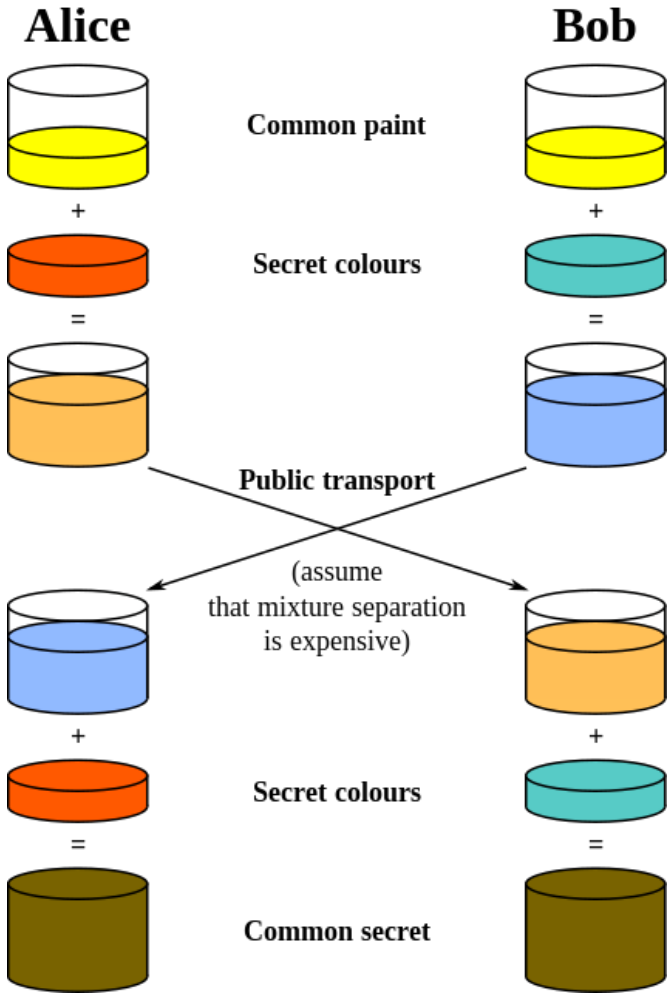
- PUBLIC
  - p = 11
  - g = 2
  - (g is a generator for group mod p)

- Alice: x=9, sends 6 (g^x mod p = 2^9 mod 11 = 6)
- Bob: y=4, send 5 (g^y mod p = 2^4 mod 11 = 5)

- A compute:  5^x mod 11 (5^9 mod 11 = 9)
- B compute 6^y mod 11 (6^4 mod 11 = 9)
- Both get 9

- All computations modulo 11

# 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]

# Why is Diffie-Hellman Secure?

- Discrete Logarithm (DL) problem:
  given $g^x \bmod p$, it's hard to extract x
  - There is no known efficient algorithm for doing this
  - This is not 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

# Diffie-Hellman Caveats (1)

- 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 $g^{xy}$ *mod p*, and use the hash as the key
  - Can use the new key for symmetric cryptography

# Diffie-Hellman Caveats (2)

- Diffie-Hellman protocol (by itself) does not provide authentication (against active attackers)
  - Person in the middle attack (aka "man in the middle attack")

# Diffie-Hellman Key Exchange Today

- 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
    - Today's de-facto standard

# 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 $\phi(n)$ ($n \geq 1$) 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: $\phi(p) = p-1$
  - Note that $\phi(ab) = \phi(a)\,\phi(b)$ 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 $\phi$**(n)**=(p-1)(q-1)
  - Choose small **e**, relatively prime to $\phi$(n)
    - Typically, **e=3** or **e=$2^{16}$+1=65537**
  - Compute unique **d** such that ed ≡ 1 mod $\phi$(n)
    - Modular inverse: d ≡ $e^{-1}$ mod $\phi$(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

How to compute?

- Extended Euclidian algorithm
- Wolfram Alpha ☺
- Brute force for small values

# Why is RSA Secure?

- RSA problem: given c, n=pq, and e such that gcd(e, $\phi(n)$)=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 $e^{th}$ 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 $p_1, \ldots, p_k$ such that $n = p_1^{e_1} p_2^{e_2} \ldots p_k^{e_k}$
- 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

# Stepping Back: Asymmetric Crypto

- Last time we saw session key establishment (Diffie-Hellman)
  - Can then use shared key for symmetric crypto
- We just saw: public key encryption
  - For confidentiality
- Next time: digital signatures
  - For authenticity