CSE 484 : Computer Security and Privacy

Cryptography
[Finish Hash Functions; Start Asymmetric Cryptography]

Winter 2021

David Kohlbrenner

dkohlbre@cs.washington.edu

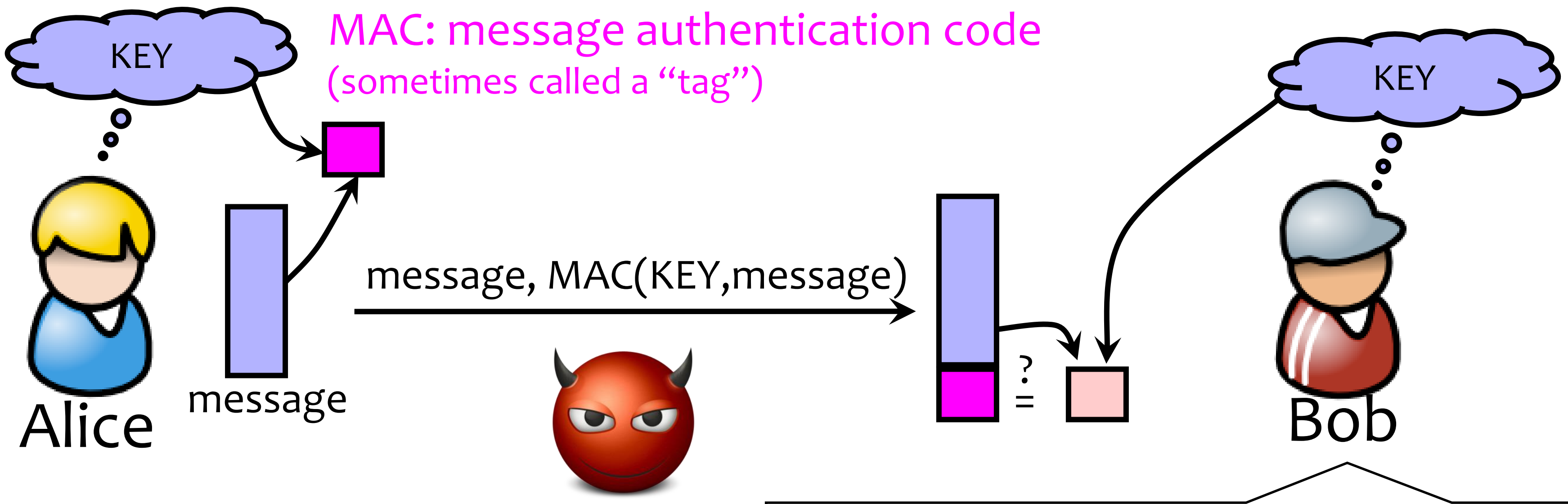
Thanks to Franzi Roesner, Dan Boneh, Dieter Gollmann, Dan Halperin, Yoshi Kohno, Ada Lerner, John Manferdelli, John Mitchell, Vitaly Shmatikov, Bennet Yee, and many others for sample slides and materials ...

Admin

- Lab 1 due on Wednesday!
  - Check your group settings on Canvas!
- Remember to do your ‘in-class’ activities, even if you watch the recordings, they are nearly free points
- Homework 2 (crypto) out now (due Feb 10)

Recall: Achieving Integrity

Message authentication schemes: A tool for protecting integrity.



Recomputes MAC and verifies whether it is equal to the MAC attached to the message

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

HMAC (older hashes)

- Construct MAC from a cryptographic hash function
  - Invented by Bellare, Canetti, and Krawczyk (1996)
  - Used in SSL/TLS, mandatory for IPsec
- Construction:
  - HMAC(k,m) = Hash((k$\oplus$ipad) || Hash(k$\oplus$opad || m))
- Why not block ciphers (at the time it was designed)?
  - 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

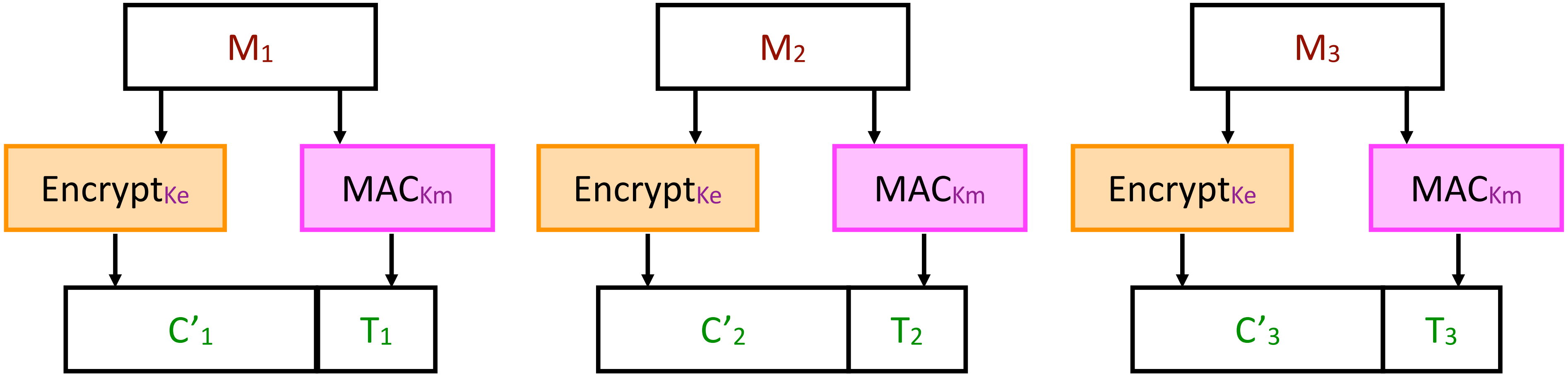
MAC with SHA3

- SHA3(Key || Message)

- SHA3 has some nice features that prevent the class of attacks HMAC prevents

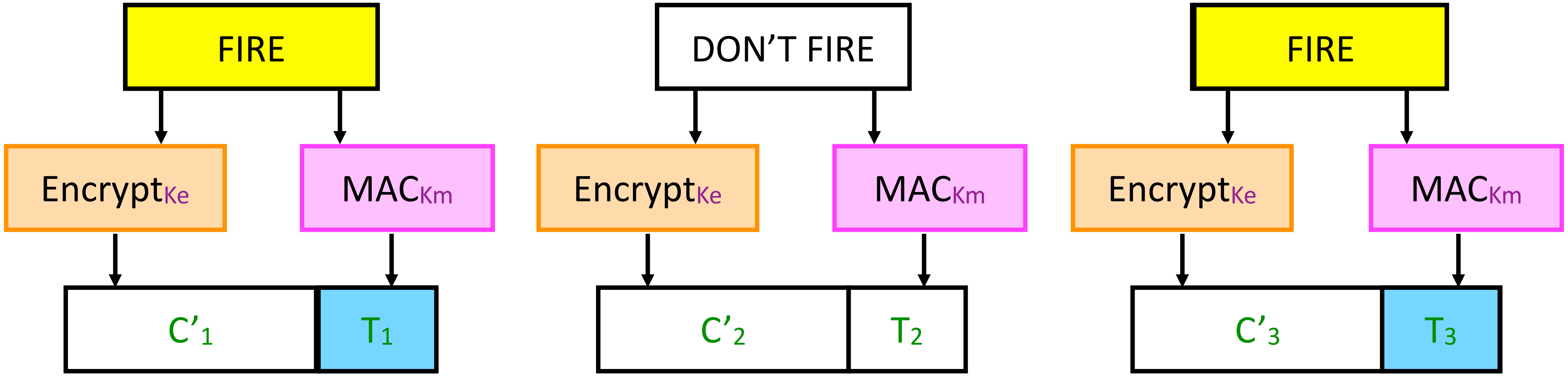
Authenticated Encryption

- What if we want both privacy and integrity?
- Natural approach: combine encryption scheme and a MAC.



Authenticated Encryption

- What if we want both privacy and integrity?
- Natural approach: combine encryption scheme and a MAC.
- But be careful!
  - Obvious approach: Encrypt-and-MAC
  - Problem: MAC is deterministic! same plaintext → same MAC

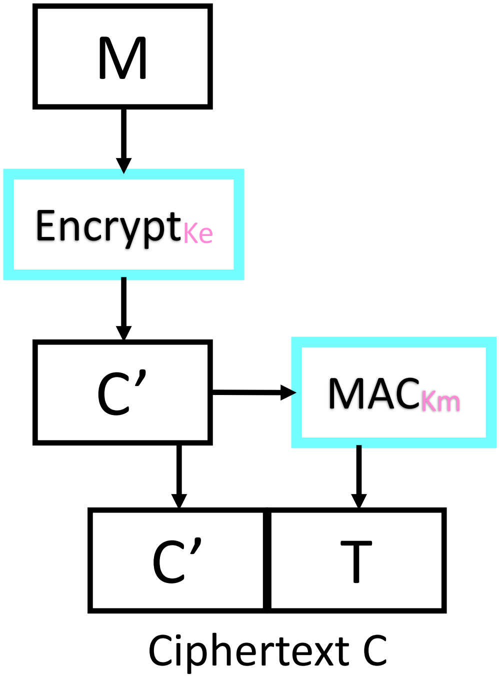


Authenticated Encryption

- Instead:

  Encrypt *then* MAC.

- (Not as good:
  MAC-then-Encrypt)



Encrypt-then-MAC

Back to cryptography land

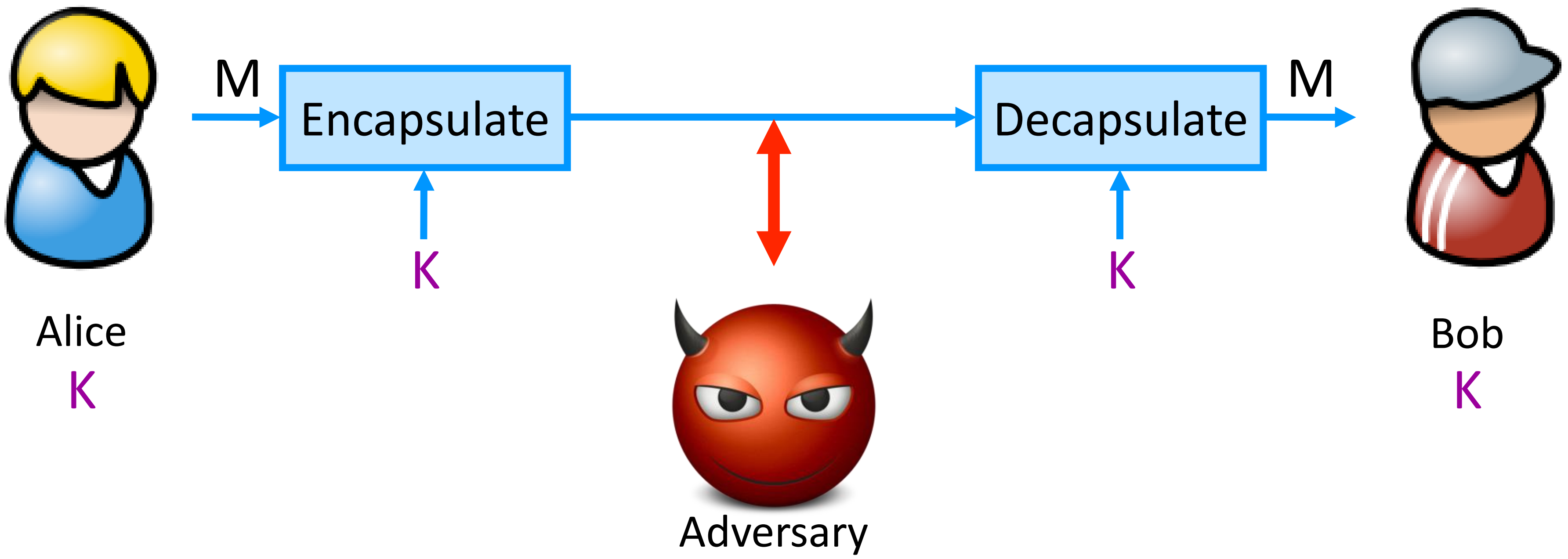
Stepping Back: Flavors of Cryptography

- 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.

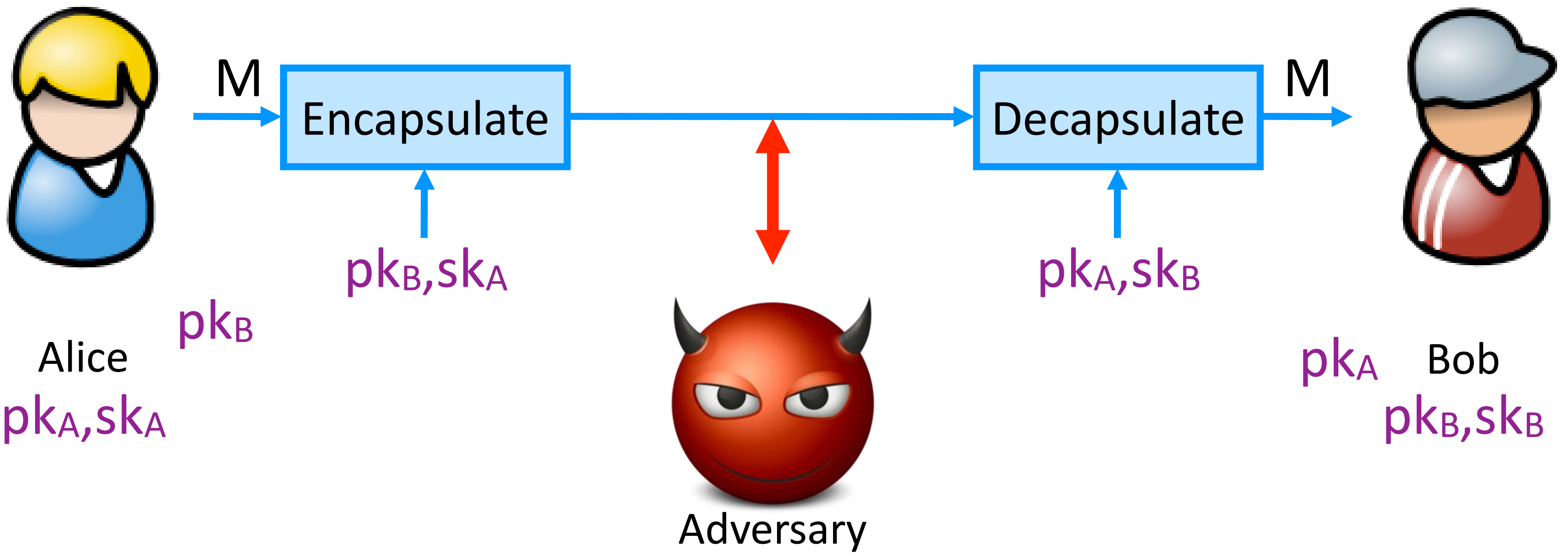
Symmetric Setting

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

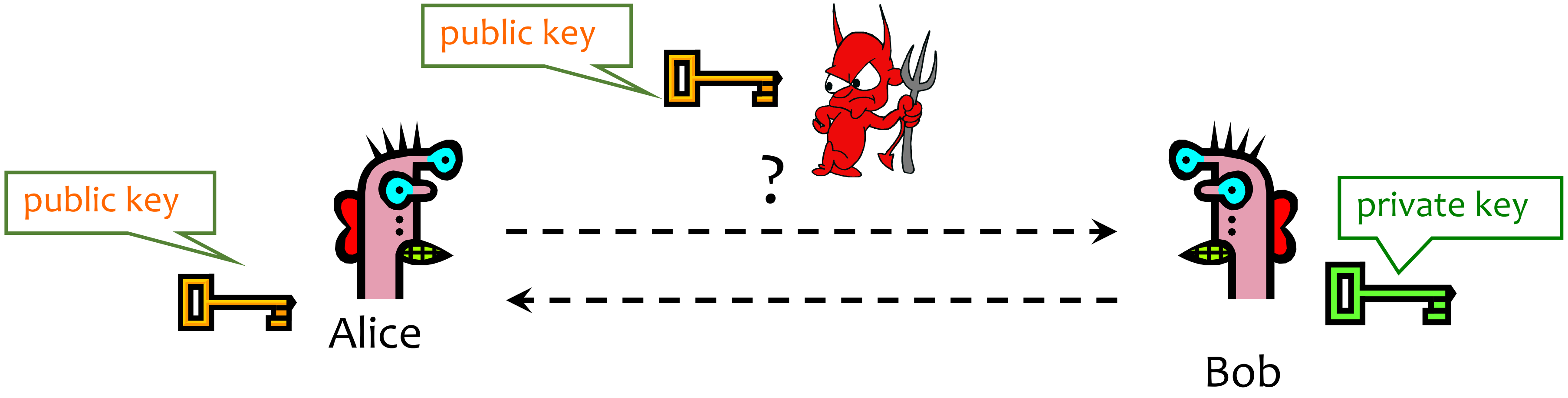


Asymmetric Setting

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 himself

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?)

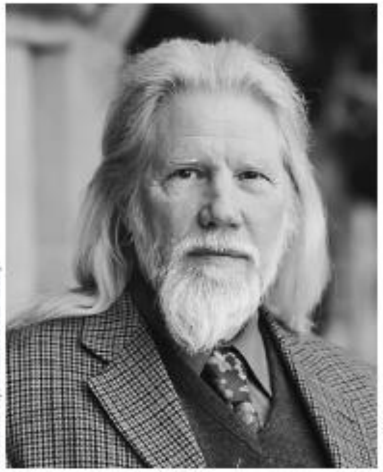
Session Key Establishment

Modular Arithmetic

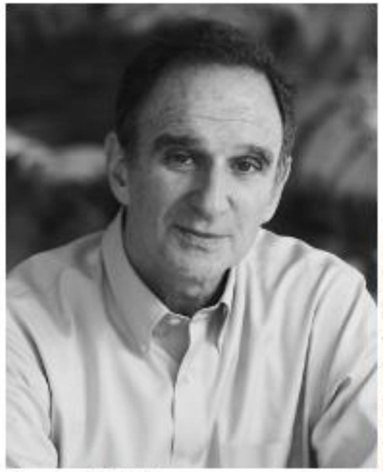
- Given g and prime p, compute: g^1 mod p, g^2 mod p, … g^{100} mod p
  - For p=11, g=10
    - 10^1 mod 11 = 10, 10^2 mod 11 = 1, 10^3 mod 11 = 10, …
    - Produces cyclic group {10, 1} (order=2)
  - For p=11, g=7
    - 7^1 mod 11 = 7, 7^2 mod 11 = 5, 7^3 mod 11 = 2, …
    - Produces cyclic group {7,5,2,3,10,4,6,9,8,1}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

Diffie-Hellman Protocol (1976)

Diffie and Hellman Receive 2015 Turing Award



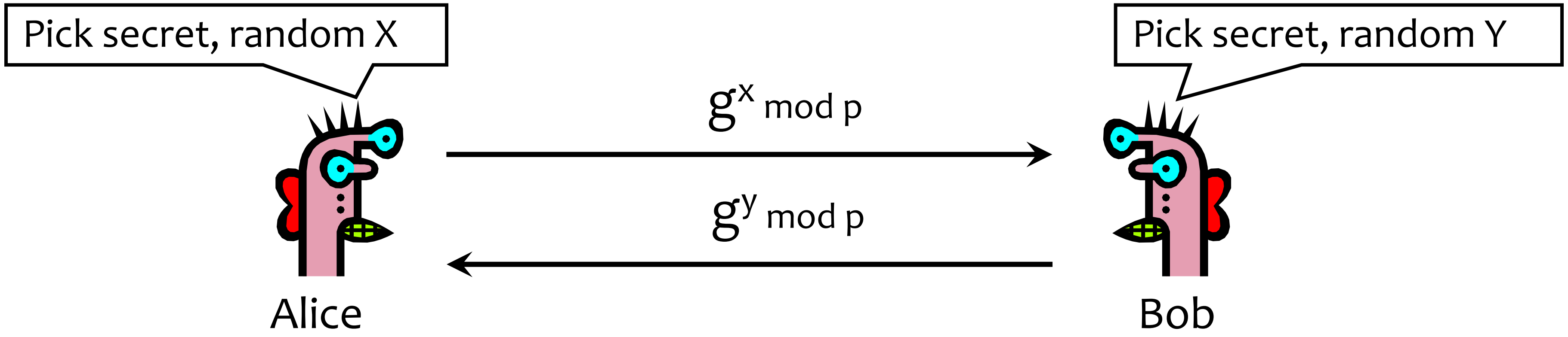
Whitfield Diffie



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^*
    - $Z_p^*=\{1, 2 \ldots p\text{-}1\}$; a Z_p^* i such that $a=g^i \bmod p$
    - Modular arithmetic: numbers “wrap around” after they reach p



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

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

Example Diffie Hellman Computation

Why is Diffie-Hellman Secure?

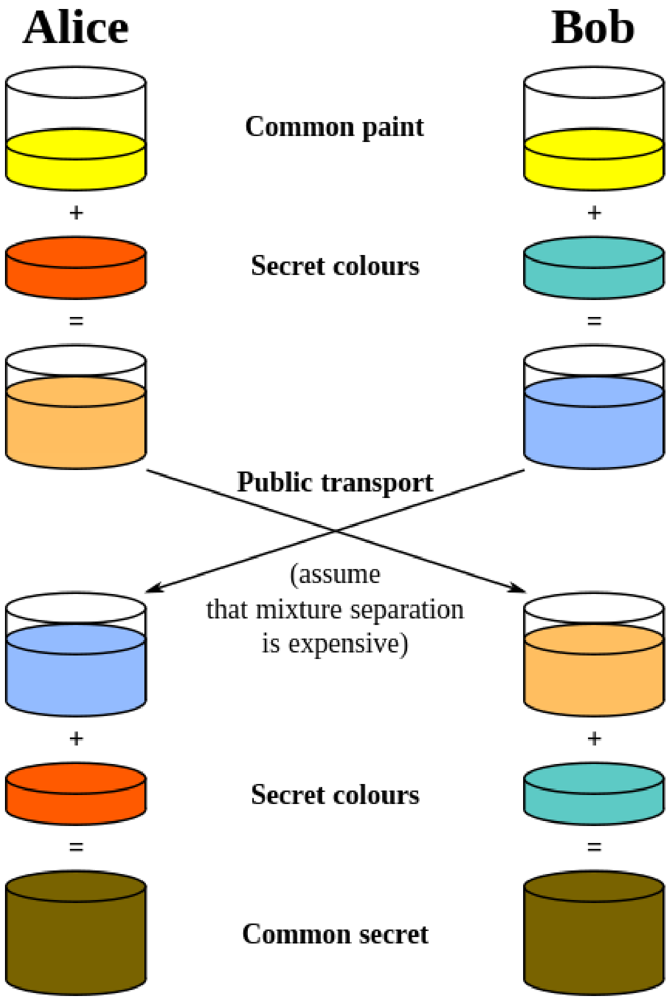
- Discrete Logarithm (DL) problem:

  given $g^x \mod p$, 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 $g^{xy} \mod p$
  - … 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 $g^{xy} \mod p$ and $g^r \mod p$ where r is random

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]

Properties of Diffie-Hellman

- 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*
  - 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 protocol (by itself) does not provide authentication (against active attackers)
  - Person in the middle attack (also called "man in the middle attack")

Person In The Middle Attack

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