CSE 484 / CSE M 584: Computer Security and Privacy

Cryptography
[Asymmetric Cryptography]

Fall 2017

Franziska (Franzi) Roesner
franzi@cs.washington.edu

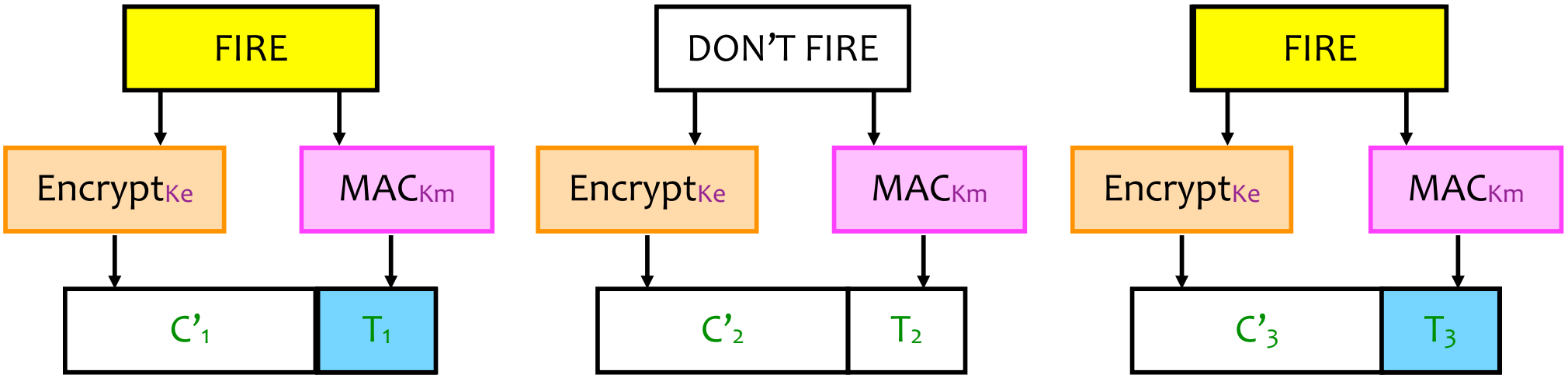
Thanks to Dan Boneh, Dieter Gollmann, Dan Halperin, Yoshi Kohno, Ada Lerner, John Manferdelli, John Mitchell, Vitaly Shmatikov, Bennet Yee, and many others for sample slides and materials ...

Announcements

- Lab #1 due today
- Coming up
  - Wednesday: tech policy (Emily McReynolds)
  - Friday: adversarial ML (Earlence Fernandes)
  - Then: web security!
- Homework #2 on crypto out on today (due 11/3)
- If office hour times don't work for you, let us know and/or schedule appointments

Recap: Authenticated Encryption

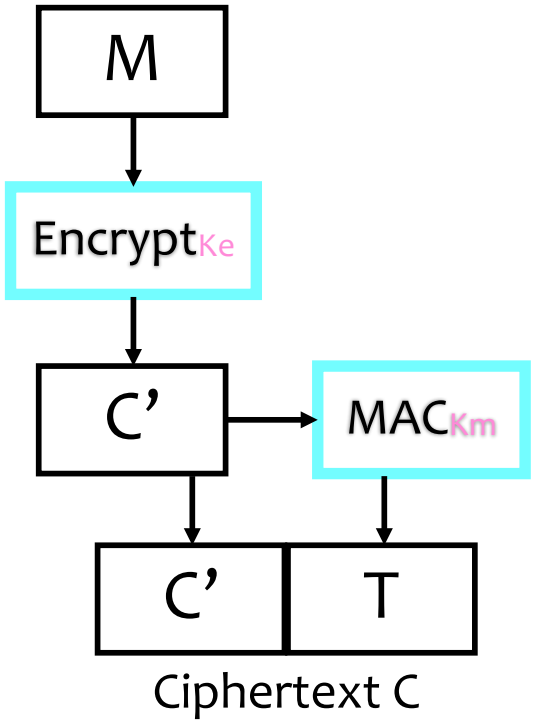
- What if we want both privacy and integrity?
- Natural approach: combine encryption scheme and a MAC.
- But be careful!
  - Obvious approach: Encrypt-and-MAC
  - Problem: MAC is deterministic! same plaintext → same MAC



Recap: Authenticated Encryption

- Instead:
  Encrypt *then* MAC.

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



Encrypt-then-MAC

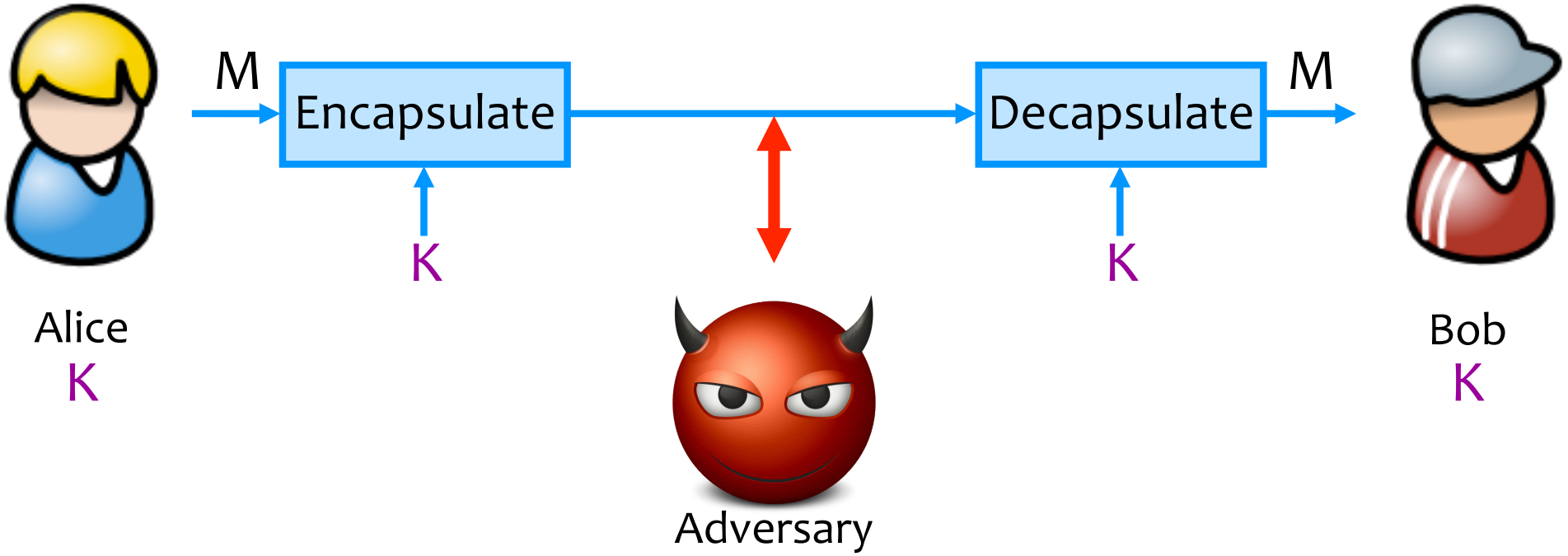
Stepping Back: Flavors of Cryptography

- 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.

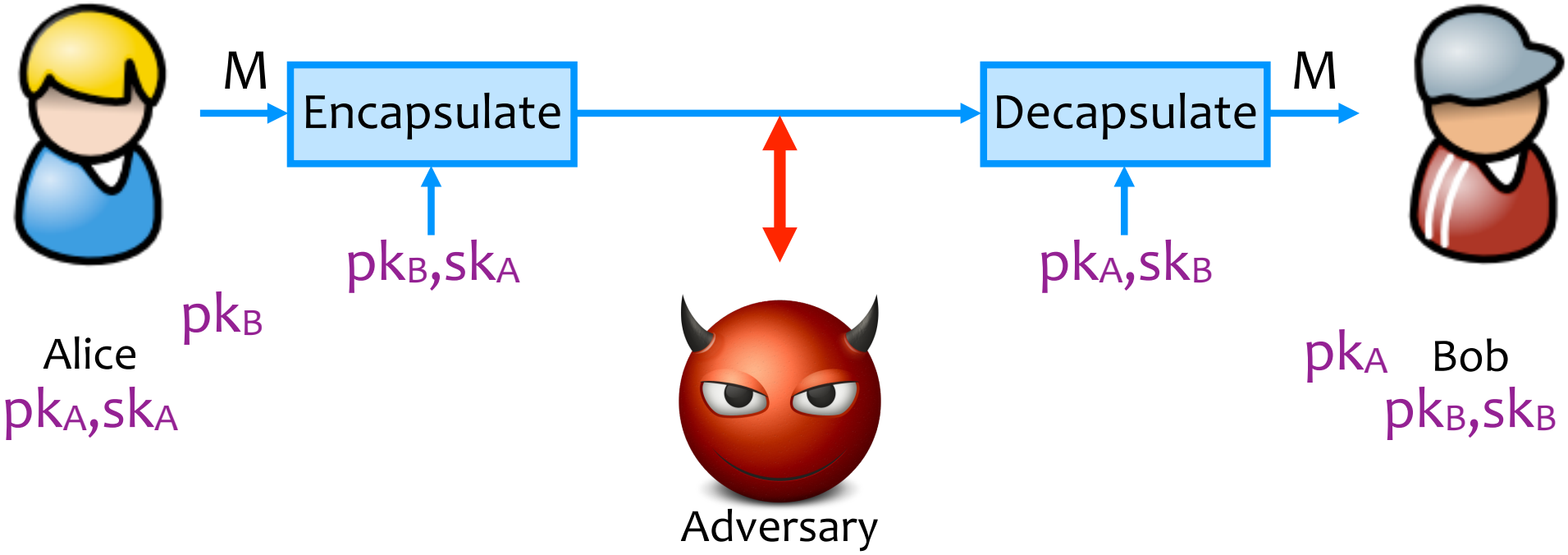
Symmetric Setting

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



Asymmetric Setting

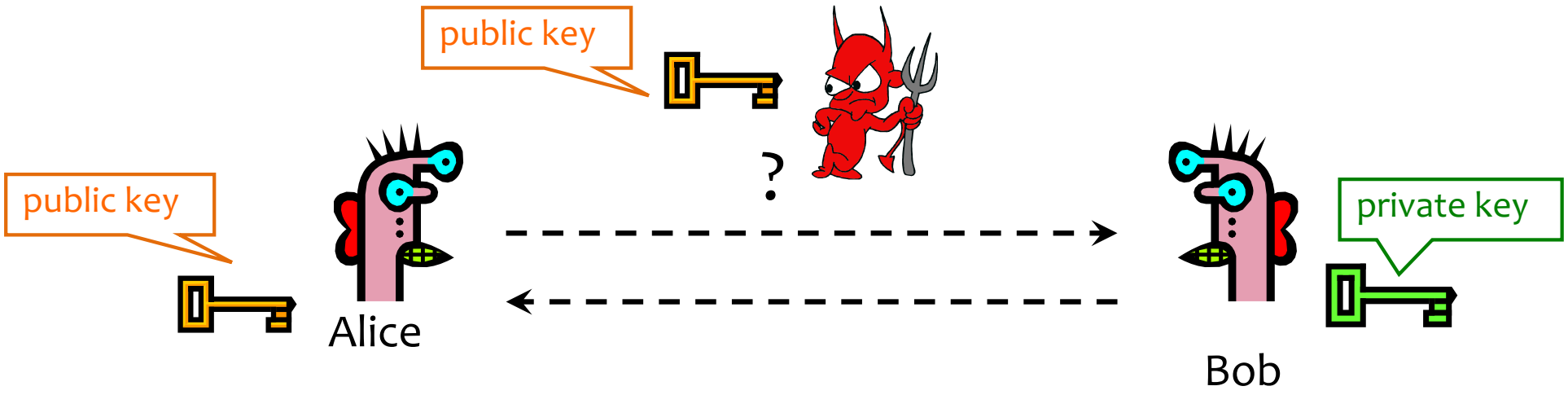
Each party creates a public key pk and a secret key sk.



Flavors of Cryptography

- Symmetric cryptography
  - Both communicating parties have access to a shared random string K, called the key.
  - Challenge: How do you privately share a key?
- Asymmetric cryptography
  - Each party creates a public key pk and a secret key sk.
  - Challenge: How do you validate a public key?

Public Key Crypto: Basic Problem



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

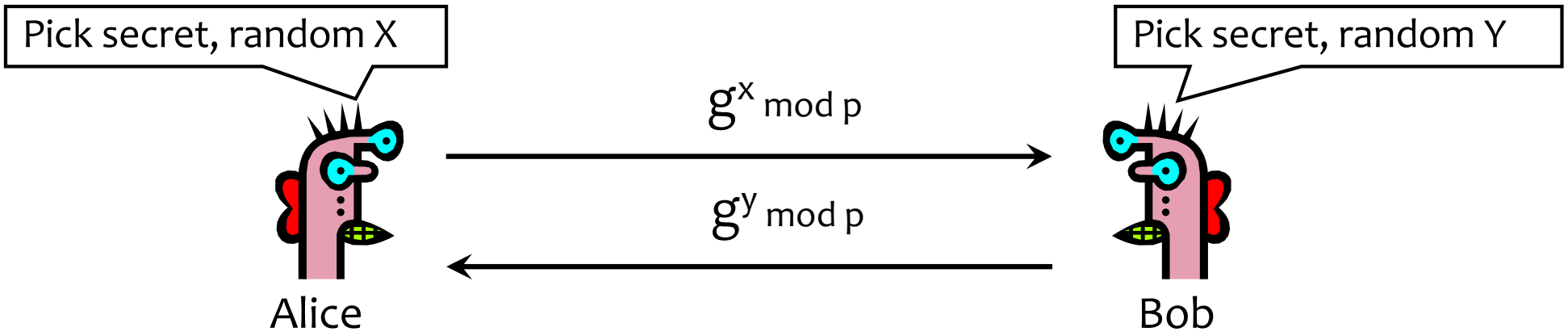
- Encryption for confidentiality
  - Anyone can encrypt a message
    - With symmetric crypto, must know secret key to encrypt
  - Only someone who knows private key can decrypt
  - Key management is simpler (or at least different)
    - Secret is stored only at one site: good for open environments
- 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?)

Modular Arithmetic

- Refresher in section last week
- Given g and prime p, compute:
  $g^1 \bmod p$, $g^{100} \bmod p$, … $g^{100} \bmod p$
  - 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)

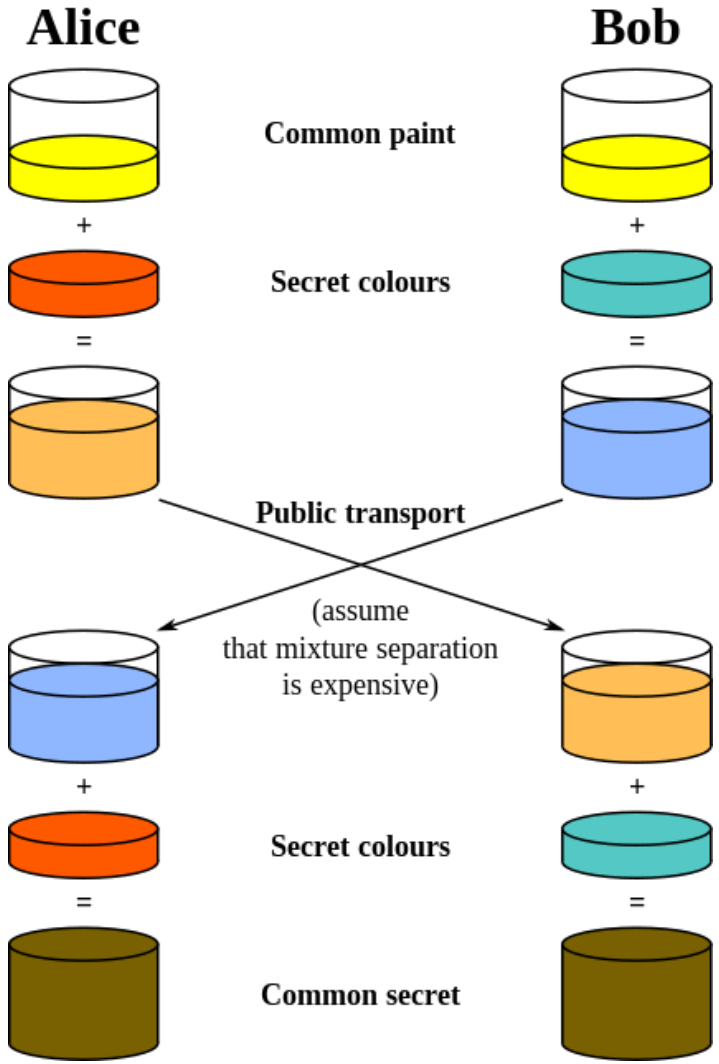
- Alice and Bob never met and share no secrets
- Public info: p and g
  - p is a large prime, g is a **generator** of Z_p^*
    - $Z_p^* = \{1, 2 \ldots p-1\}$; $\forall a \in Z_p^* \ \exists i$ such that $a = g^i \bmod p$
    - Modular arithmetic: numbers "wrap around" after they reach p



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

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

Diffie-Hellman: Conceptually



Common paint: p and g

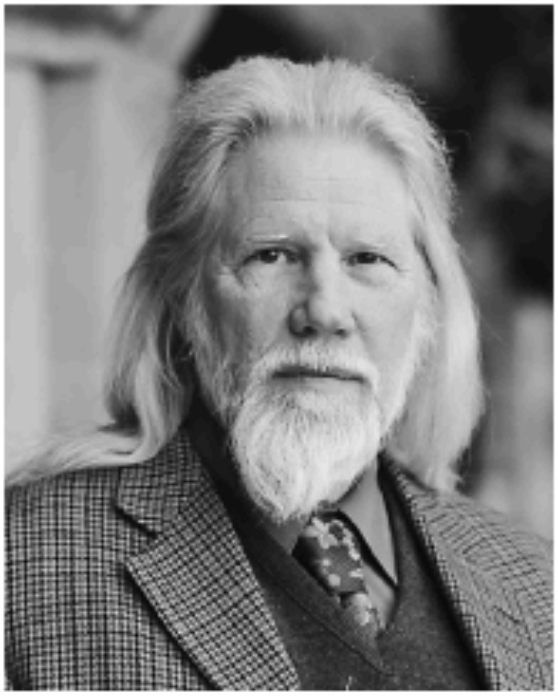
Secret colors: x and y

Send over public transport:
$g^x \bmod p$
$g^y \bmod p$

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

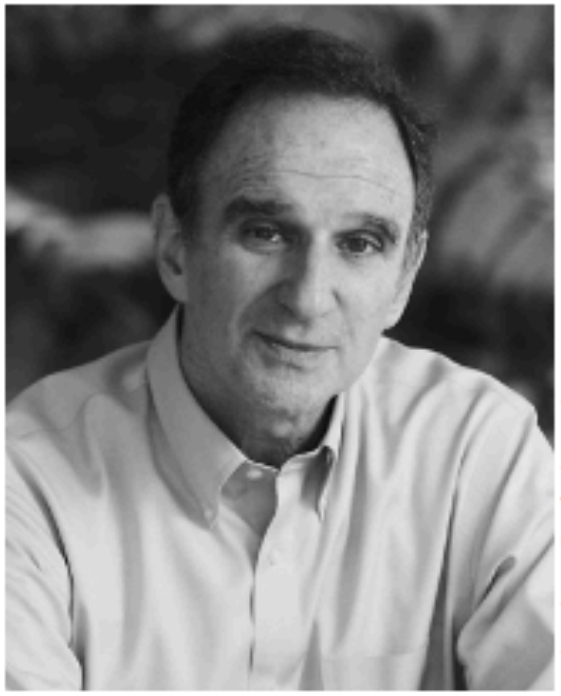
[from Wikipedia]

Diffie and Hellman Receive 2015 Turing Award



Rod Searcey/Stanford University.

Whitfield Diffie



Linda A. Cicero/Stanford News Service.

Martin E. Hellman

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

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
  - Can use the new key for symmetric cryptography
- Diffie-Hellman protocol (by itself) does not provide authentication
  - Man in the middle attack

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 $\varphi(ab) = \varphi(a)\,\varphi(b)$

RSA Cryptosystem [Rivest, Shamir, Adleman 1977]

- Key generation:
  - Generate large primes p, q
    - Say, 1024 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

How to compute?

Why is RSA Secure?

- RSA problem: given c, n=pq, and e such that gcd(e, φ(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
- 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

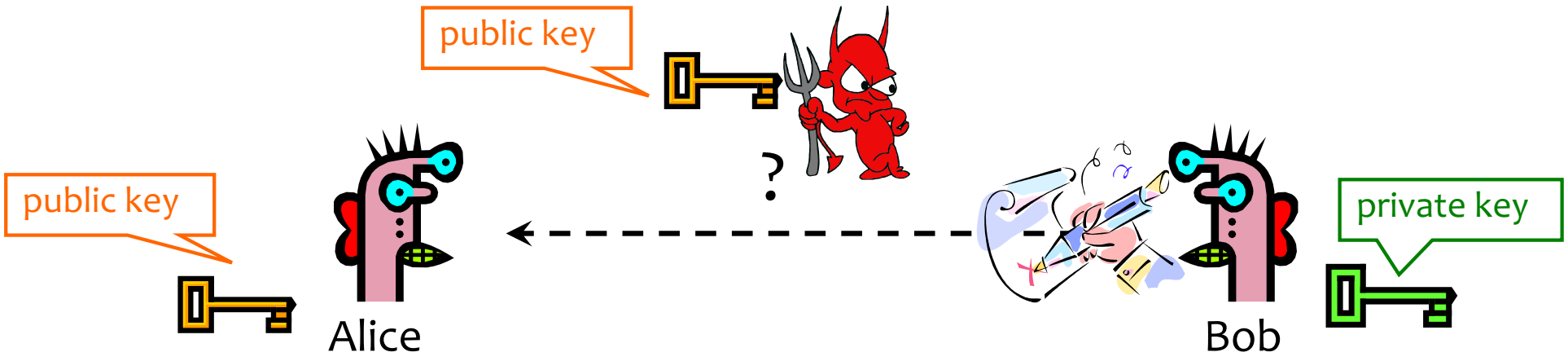
RSA Encryption Caveats

- 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

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: $s = m^d \bmod n$
  - 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 $s^e \bmod n = (m^d)^e \bmod n = m$
  - 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
  - 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: $(p, q, g, y=g^x \bmod p)$, private key: x
- Security of DSS requires hardness of discrete log
  - If could solve discrete logarithm problem, would extract x (private key) from $g^x \bmod p$ (public key)

Cryptography Summary

- Goal: Privacy
  - Symmetric keys:
    - One-time pad, Stream ciphers
    - Block ciphers (e.g., DES, AES) → modes: EBC, CBC, CTR
  - Public key crypto (e.g., Diffie-Hellman, RSA)
- Goal: Integrity
  - MACs, often using hash functions (e.g, MD5, SHA-256)
- Goal: Privacy and Integrity
  - Encrypt-then-MAC
- Goal: Authenticity (and Integrity)
  - Digital signatures (e.g., RSA, DSS)