CSE 484 / CSE M 584: Computer Security and Privacy

Cryptography [Finish Hash Functions; Start Asymmetric Cryptography]

Spring 2020

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 ...

Admin

- Lab 1 due in a week
- Homework 2 (crypto) out now (due May 8)
- Looking ahead:
  - **Today+Monday:** Asymmetric Crypto
  - **Monday:** Start transition to web security
    - Lab 2 will be on web security

Which Property Do We Need?

- 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
- Private auction bidding
  - Alice wants to bid B, sends H(B), later reveals B
  - **One-wayness:** rival bidders should not recover B (this may mean that she needs to hash some randomness with B too)
  - **Collision resistance:** Alice should not be able to change her mind to bid B' such that H(B)=H(B')

Common Hash Functions

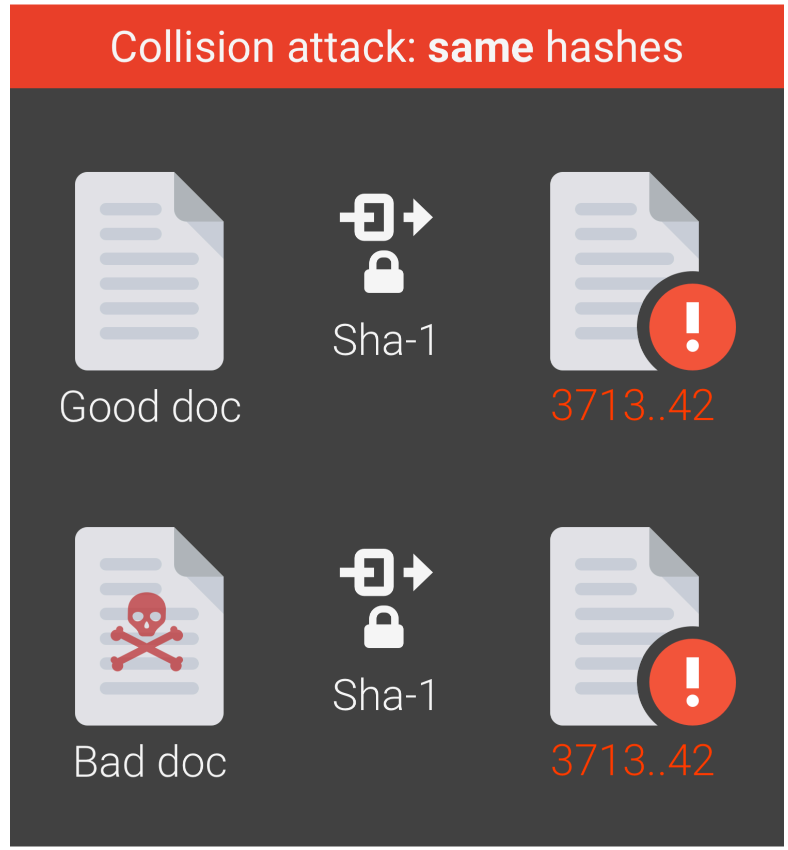
- MD5 – Don't Use!
  - 128-bit output
  - Designed by Ron Rivest, used very widely
  - Collision-resistance broken (summer of 2004)
- RIPEMD-160
  - 160-bit variant of MD5
- SHA-1 (Secure Hash Algorithm)
  - 160-bit output
  - US government (NIST) standard as of 1993-95
  - Theoretically broken 2005;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

SHA-1 Broken in Practice (2017)

Google just cracked one of the building blocks of web encryption (but don't worry)

It's all over for SHA-1

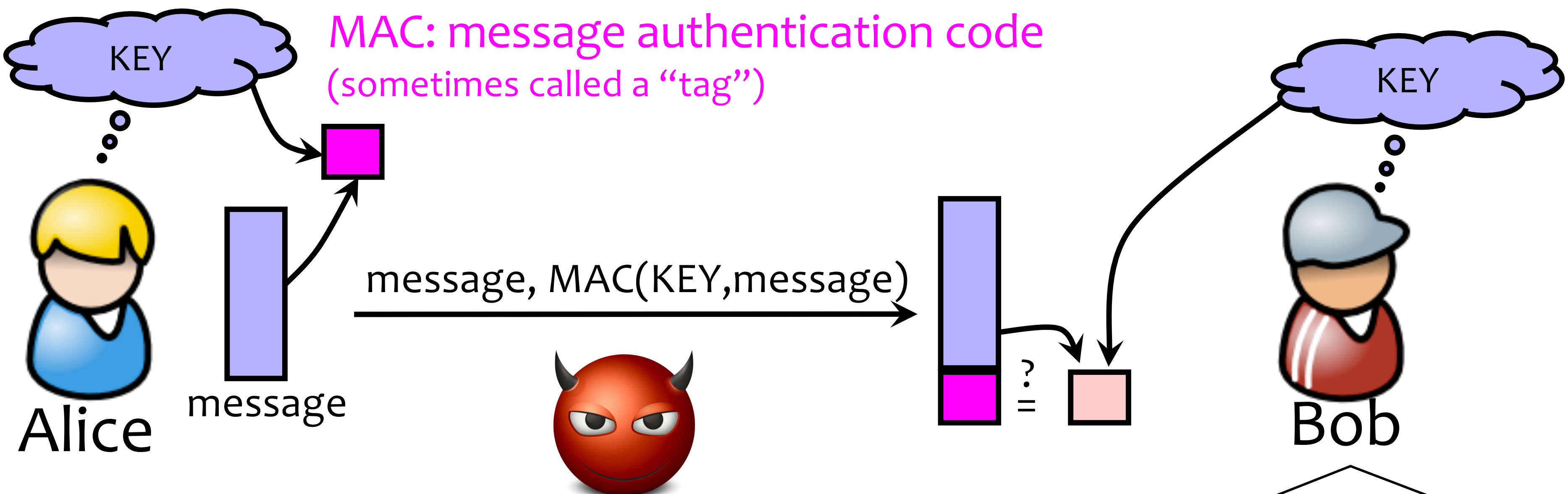
by Russell Brandom | @russellbrandom | Feb 23, 2017, 11:49am EST



https://shattered.io

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

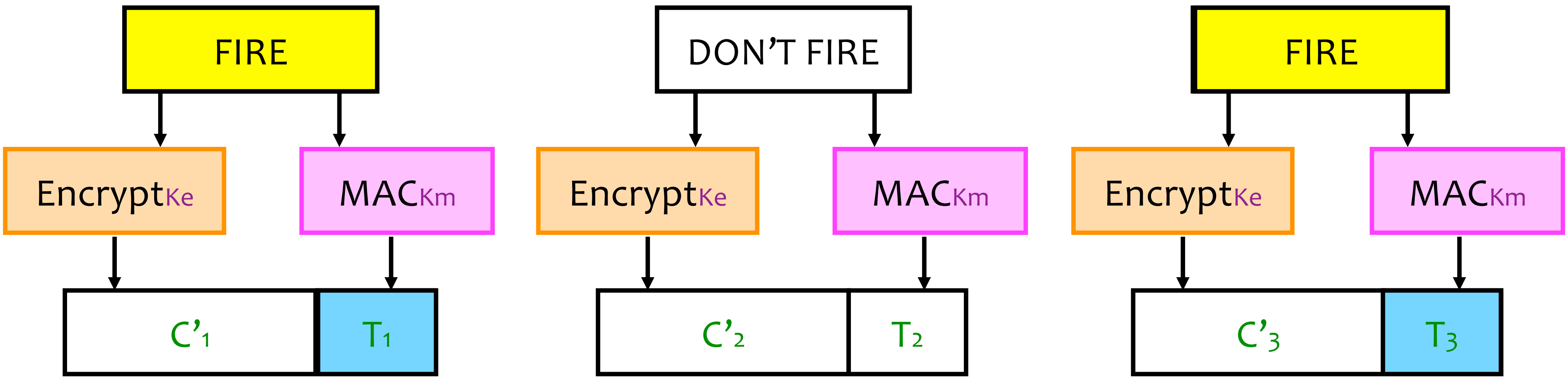
Integrity and authentication: only someone who knows KEY can compute correct MAC for a given message.

HMAC

- 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⊕ipad) | Hash(k⊕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

Authenticated Encryption

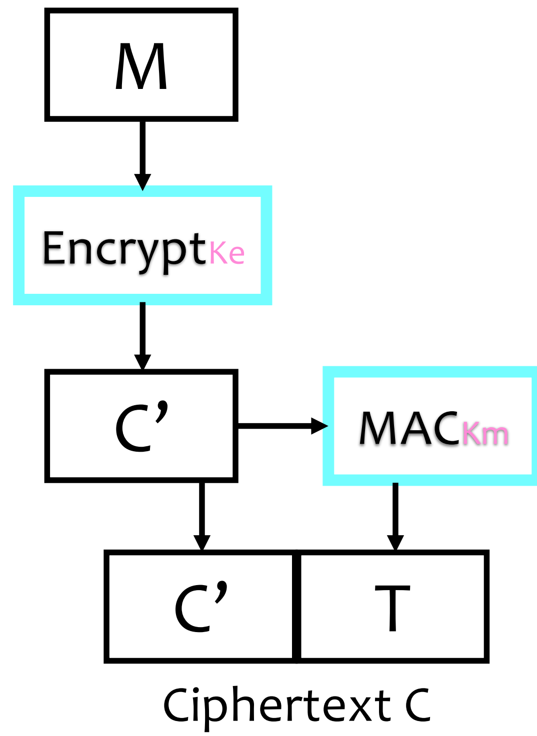
- What if we want both privacy and integrity?
- Natural approach: combine encryption scheme and a MAC.
- But be careful!
  - Obvious approach: Encrypt-and-MAC
  - Problem: MAC is deterministic! same plaintext → same MAC



Authenticated Encryption

- Instead:
  Encrypt *then* MAC.

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



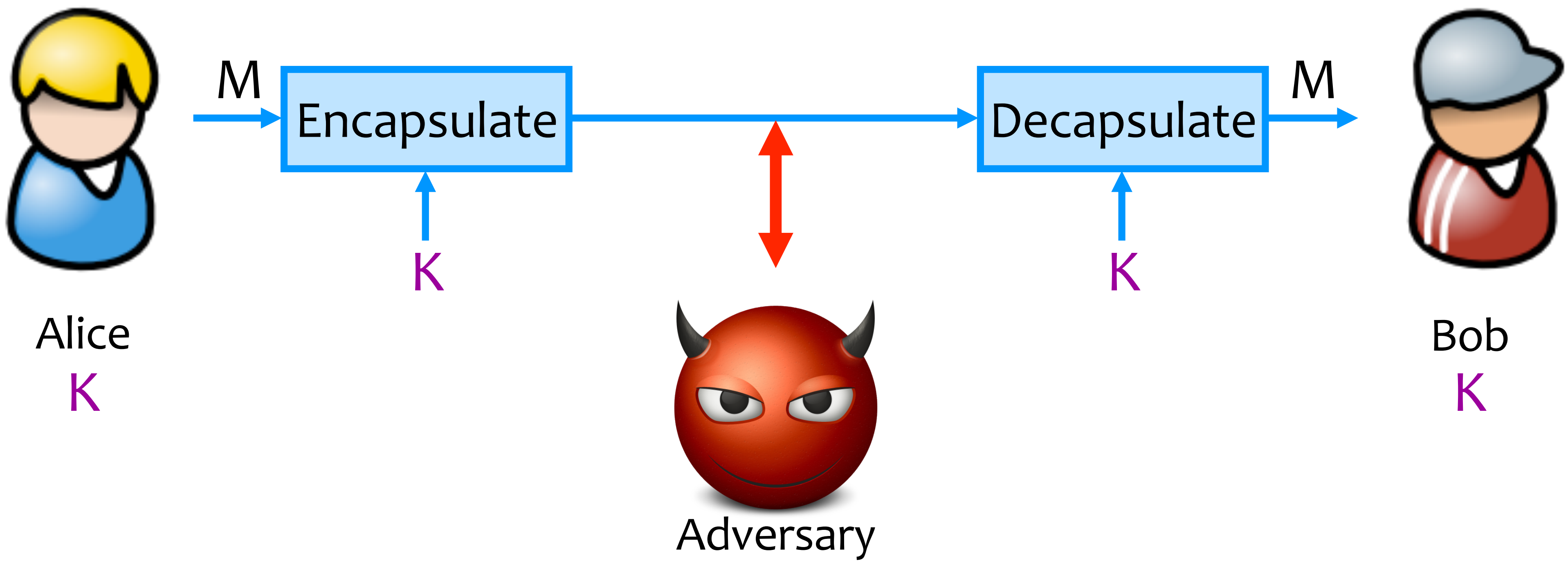
Encrypt-then-MAC

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.

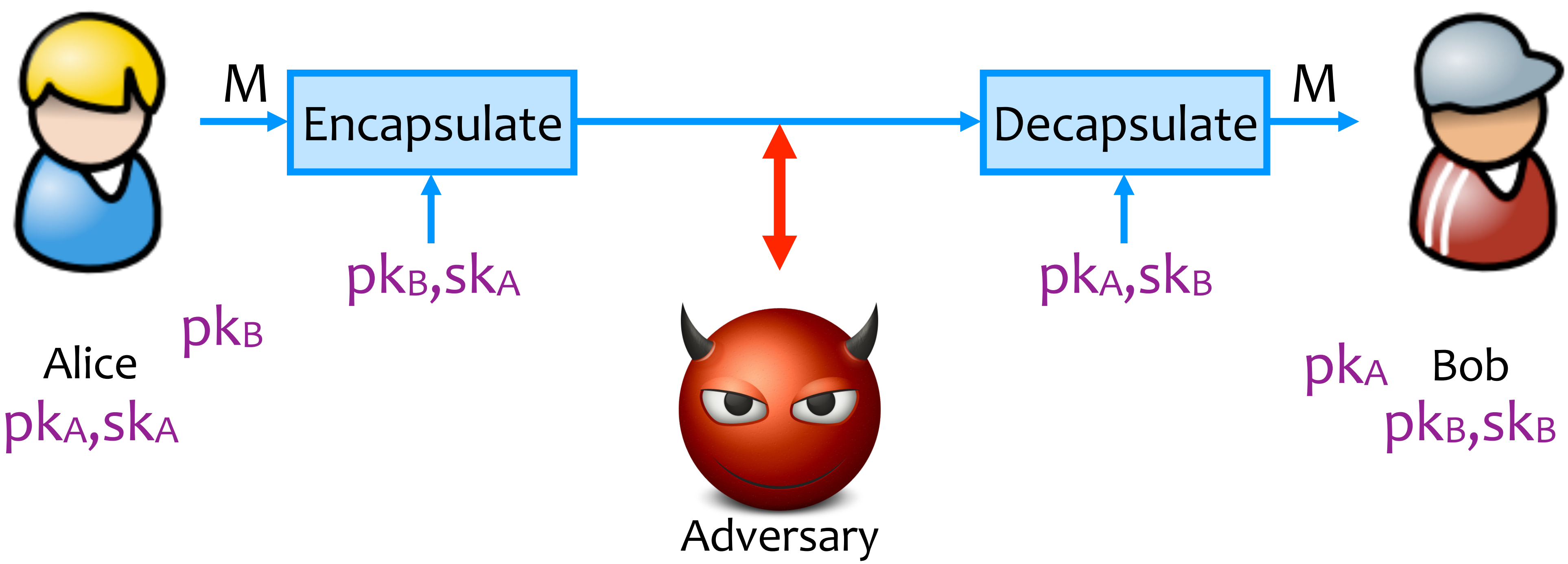
Symmetric Setting

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

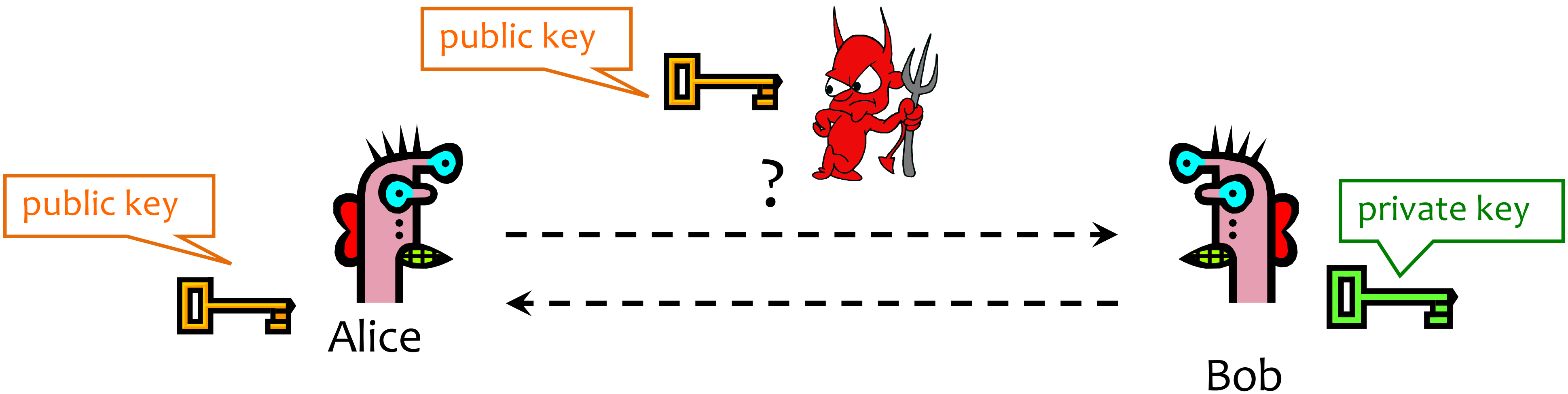


Asymmetric Setting

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?)

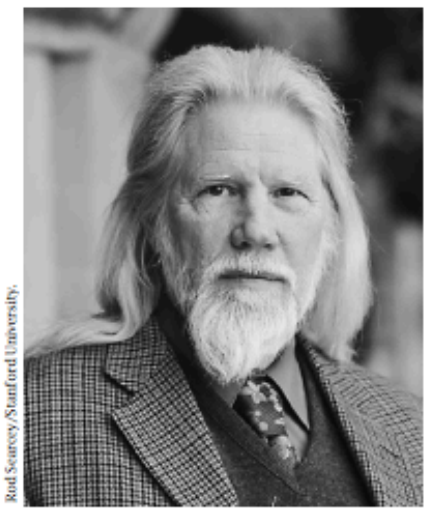
Session Key Establishment

Modular Arithmetic

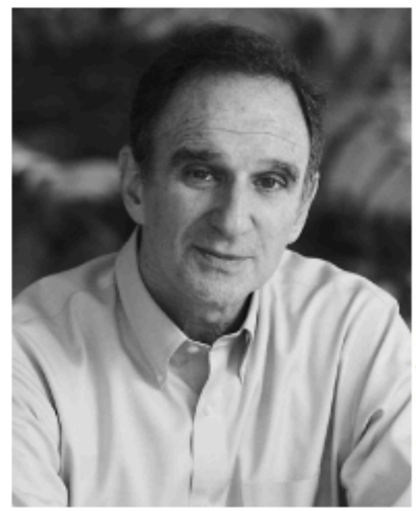
- Refresher in section yesterday
- Given g and prime p, compute:
  g^1 mod p, g^2 mod p, … g^{100} mod p
  - For p=11, g=10
    - 10^1 mod 11 = 10, 10^2 mod 11 = 1, 10^3 mod 11 = 10, …
    - Produces cyclic group {10, 1} (order=2)
  - For p=11, g=7
    - 7^1 mod 11 = 7, 7^2 mod 11 = 5, 7^3 mod 11 = 2, …
    - 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



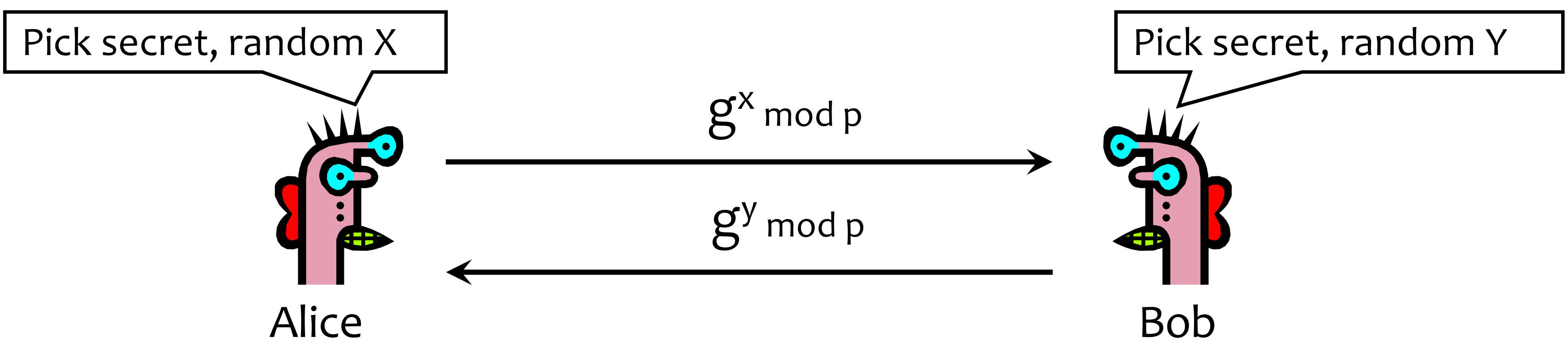
Whitfield Diffie



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*
    - $Z_p^* = \{1, 2 \ldots p-1\}$; $\forall a \in Z_p^* \;\; \exists i$ 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 $k=(g^x)^y=g^{xy} \bmod p$

Example Diffie Hellman Computation