CSE 484 / CSE M 584: Applied Cryptography

Winter 2023

Tadayoshi (Yoshi) Kohno
yoshi@cs.Washington.edu

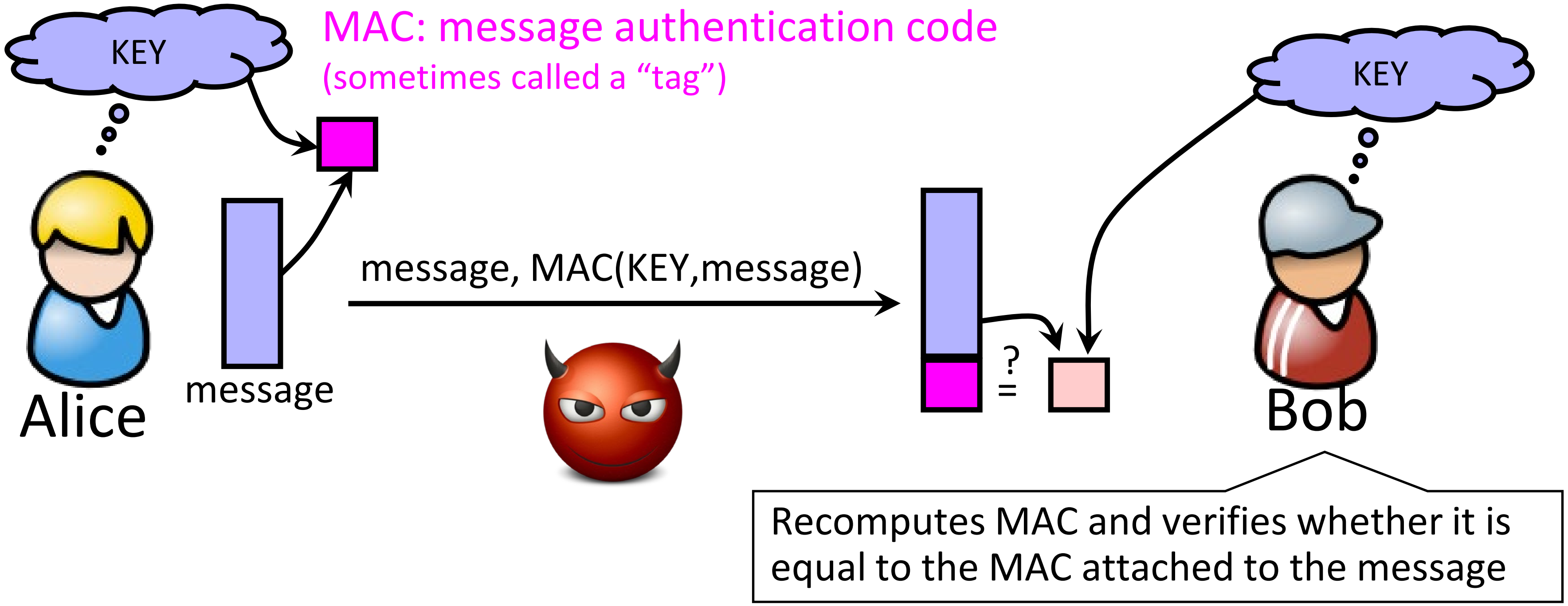
UW Instruction Team: David Kohlbrenner, Yoshi Kohno, Franziska Roesner. Thanks to Dan Boneh, Dieter Gollmann, Dan Halperin, John Manferdelli, John Mitchell, Vitaly Shmatikov, Bennet Yee, and many others for sample slides and materials ...

Announcements / Plan

- Friday (2/3) through Wednesday (2/8): Applied Crypto
- Friday (2/10): Guest Lecture: Prof. Elissa Redmiles (MPI)
- Wednesday (2/22): At most Zoom
- Friday (2/24): Guest Lecture: Alex Gantman (Qualcomm) (On Zoom)

Review: Now: Achieving Integrity

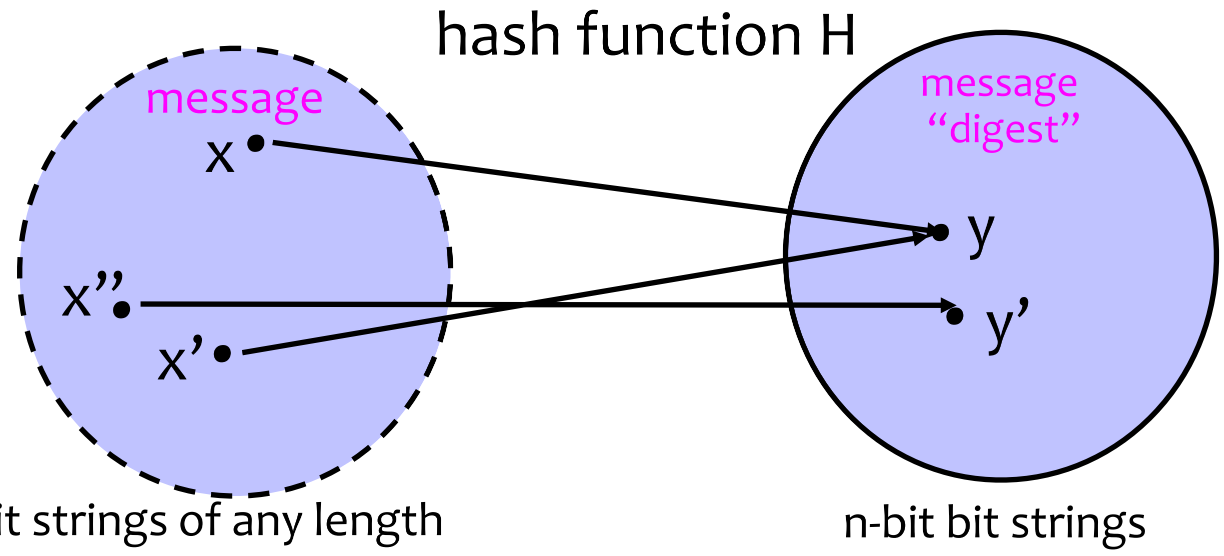
Message authentication schemes: A tool for protecting integrity.



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

Another Tool: Hash Functions

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…

Property 1: One-Way

- Intuition: hash should be hard to invert
  - "Preimage resistance"
  - Let h(x') = y in $\{0,1\}^n$ for a random x'
  - Given y, it should be hard to find any x such that h(x)=y
- How hard?
  - Brute-force: try every possible x, see if h(x)=y
  - SHA-1 (common hash function) has 160-bit output
    - Expect to try 2^{159} inputs before finding one that hashes to y.

Property 2: Collision Resistance

- Should be hard to find $x \neq x'$ such that $h(x)=h(x')$

Birthday Paradox

- Are there two people in the first 1/8 of this class that have the same birthday?
  - 365 days in a year (366 some years)
    - Pick one person. To find another person with same birthday would take on the order of 365/2 = 182.5 people
    - **Expect birthday "collision" with a room of only 23 people.**
    - For simplicity, approximate when we expect a collision as **sqrt(365)**.
- Why is this important for cryptography?
  - 2^{128} different 128-bit values
    - Pick one value at random. To exhaustively search for this value requires trying on average 2^{127} values.
    - **Expect "collision" after selecting approximately 2^{64} random values.**
    - **64 bits** of security against collision attacks, not 128 bits.

Property 2: Collision Resistance

- Should be hard to find $x \neq x'$ such that $h(x)=h(x')$
- Birthday paradox means that brute-force collision search is only $O(2^{n/2})$, *not* $O(2^n)$
  - For SHA-1, this means $O(2^{80})$ vs. $O(2^{160})$

One-Way vs. Collision Resistance

One-wayness does **not** imply collision resistance.

Collision resistance does **not** imply one-wayness.

One can prove this by constructing a function that has one property but not the other.

Property 3: Weak Collision Resistance

- Given randomly chosen x, hard to find x' such that h(x)=h(x')
  - Attacker must find collision for a specific x. By contrast, to break collision resistance it is enough to find any collision.
  - Brute-force attack requires $O(2^n)$ time
- Weak collision resistance does not imply collision resistance.

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?

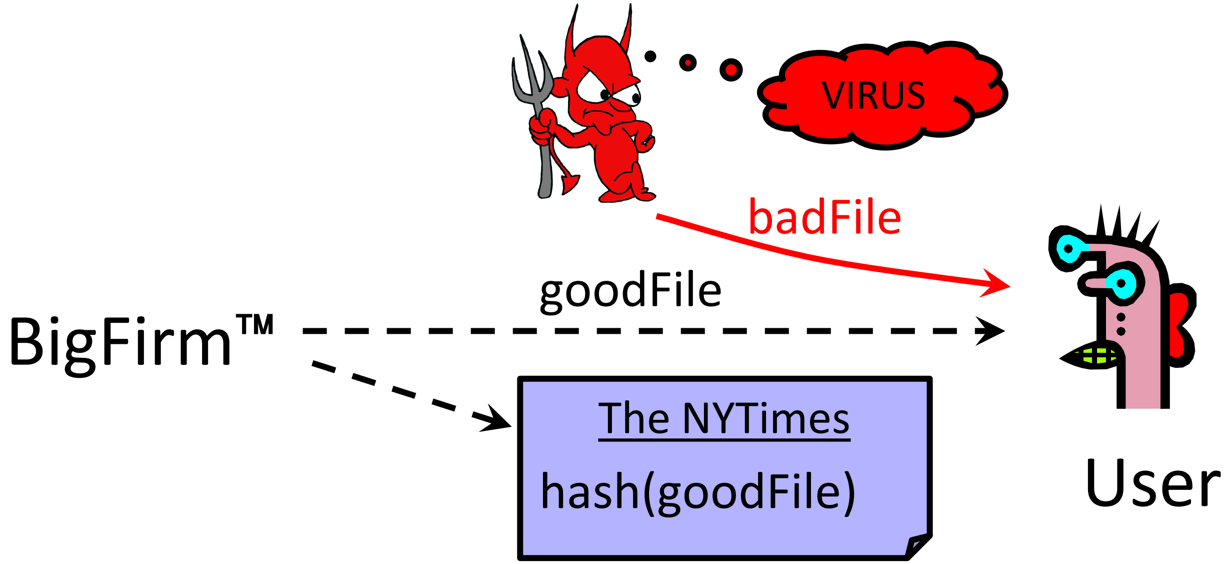
- System does not store actual passwords
- Don't need to worry about where to store the key
- Cannot go from hash to password

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!

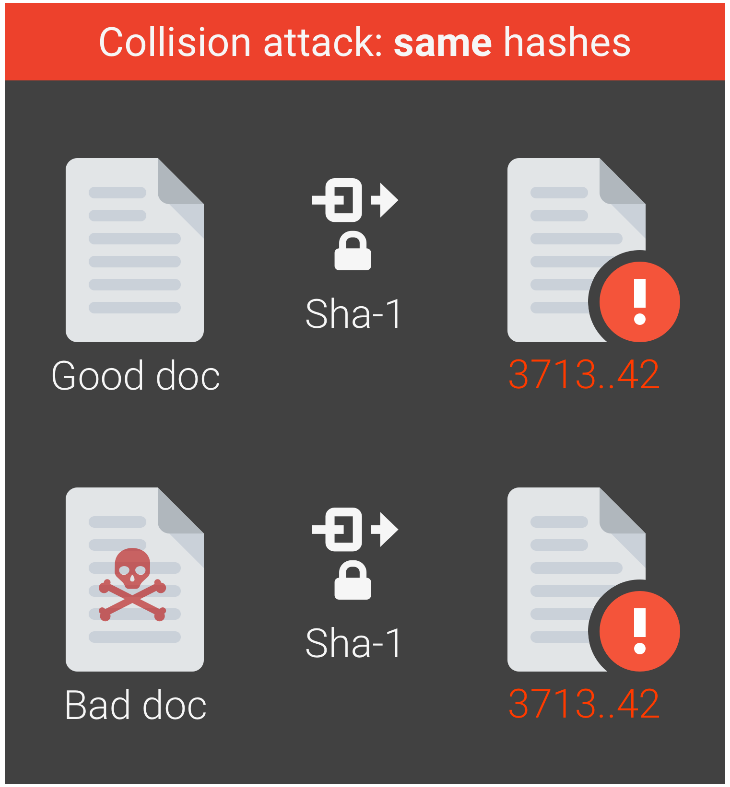
SHA-1 Broken in Practice (2017)

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

It's all over for SHA-1

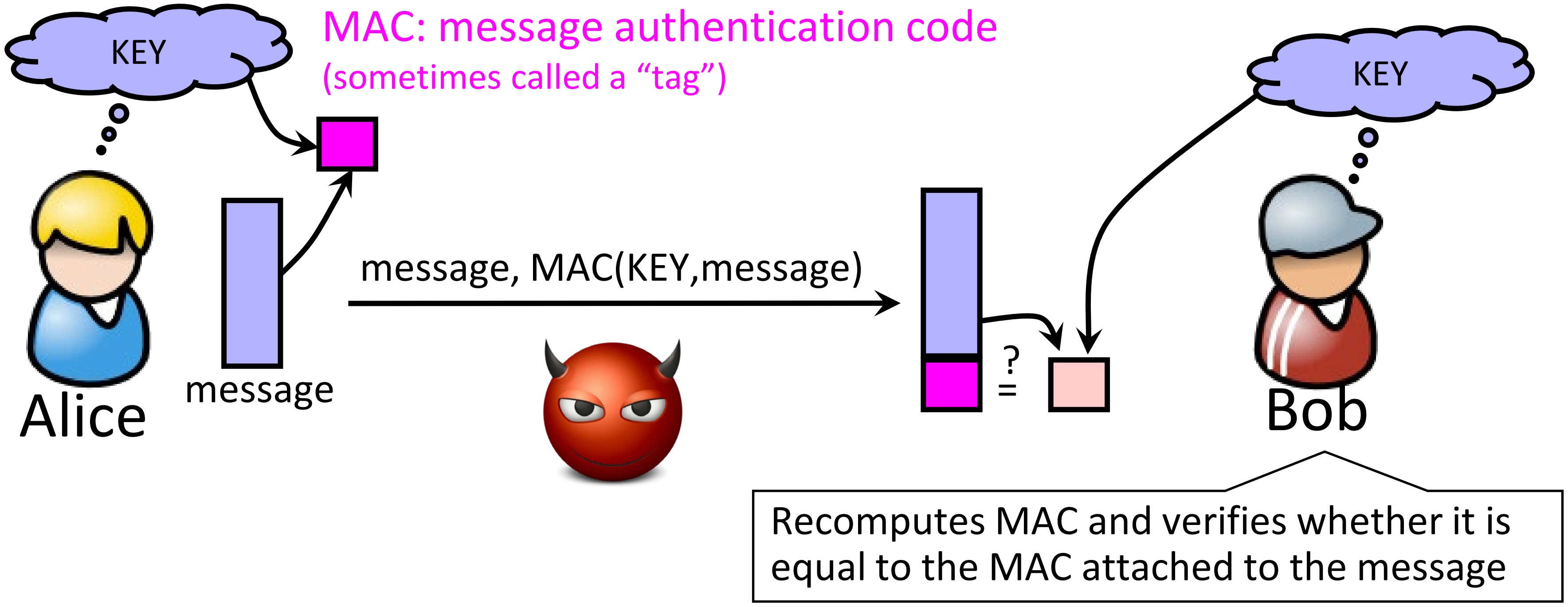
by Russell Brandom | @russellbrandom | Feb 23, 2017, 11:49am EST

https://shattered.io



Recall: Achieving Integrity

Message authentication schemes: A tool for protecting integrity.



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

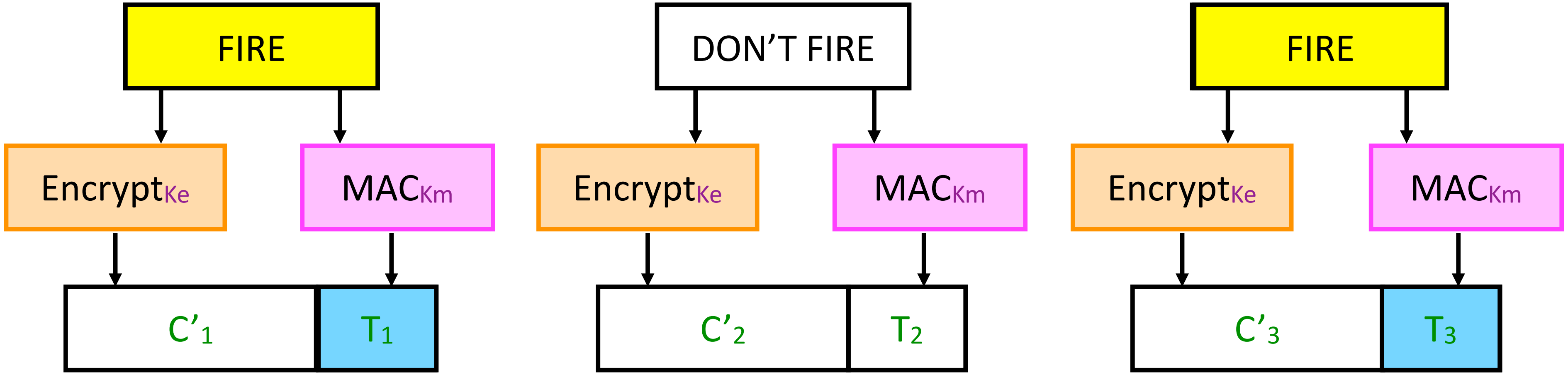
MAC with SHA3

- SHA3(Key || Message)
- Nice and simple ☺
- Previous hash functions couldn't quite be used in this way (see: length extension attack)
  - HMAC construction (FYI), roughly H(K1,H(K2,M))

- 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

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

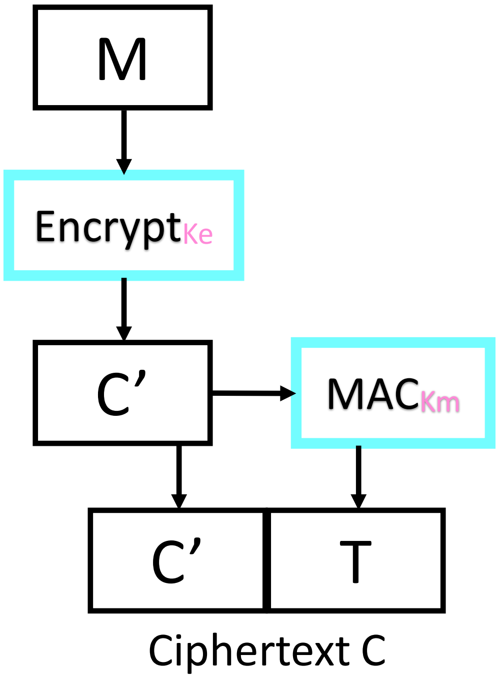


Authenticated Encryption

- Instead:

  Encrypt *then* MAC.

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



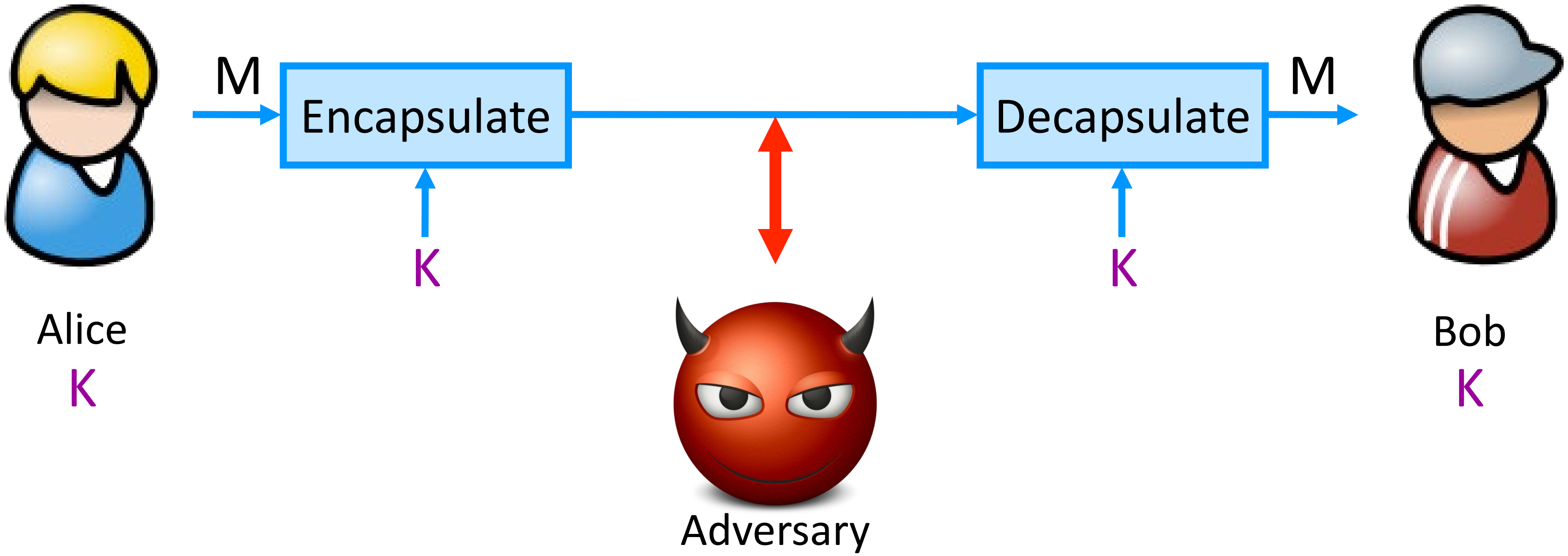
Encrypt-then-MAC

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.

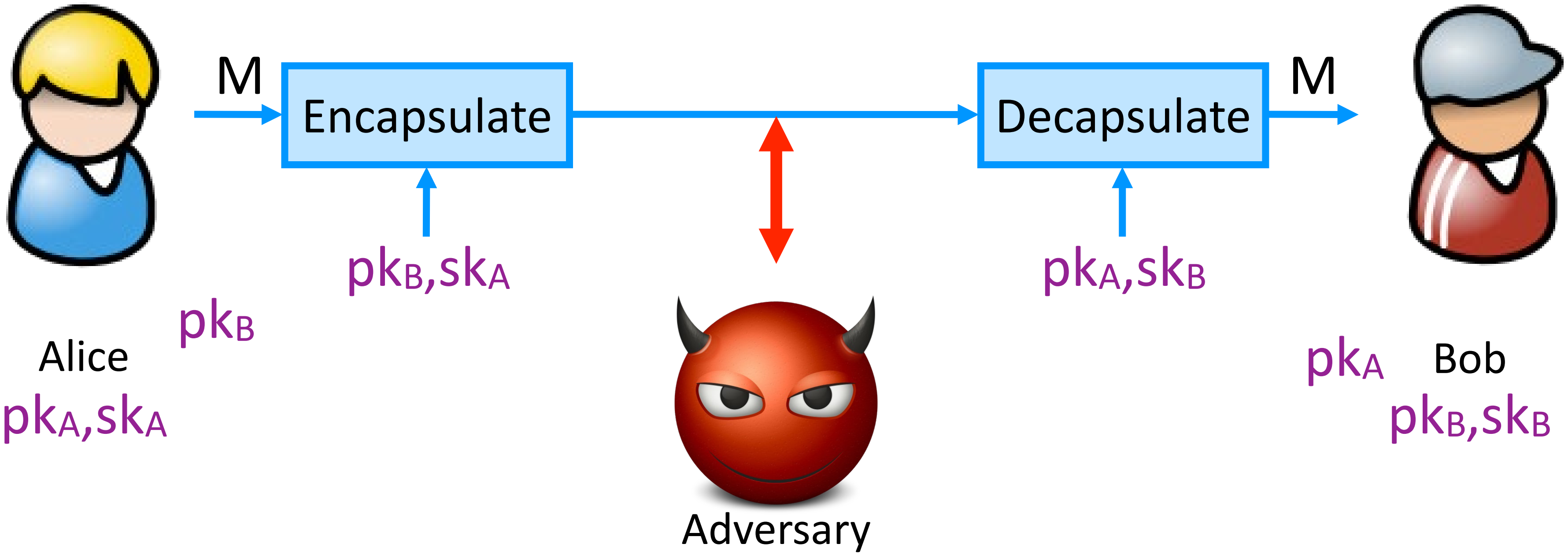
Symmetric Setting

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

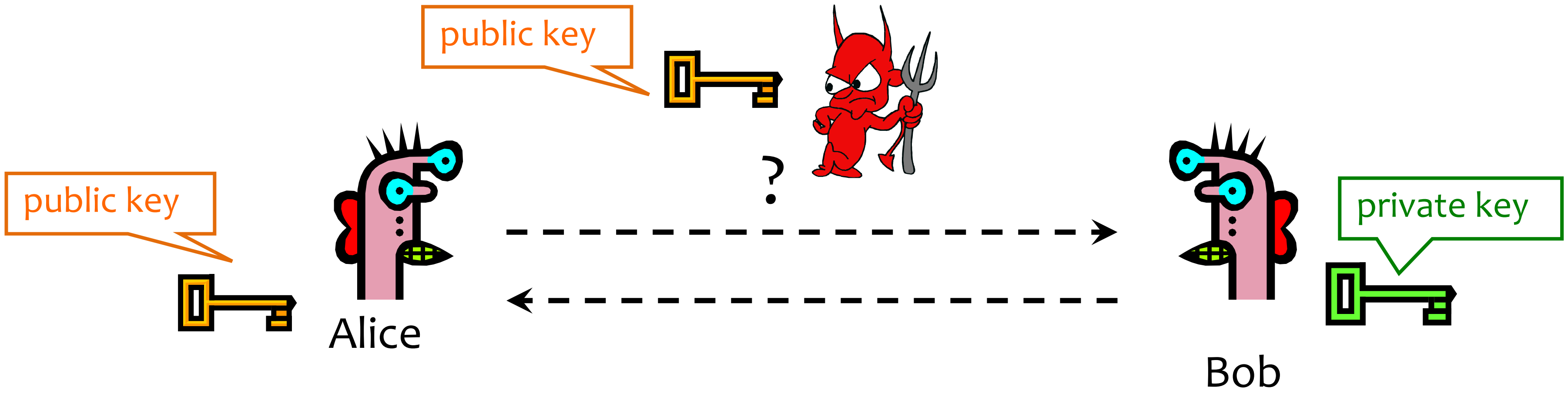


Asymmetric Setting

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 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?)

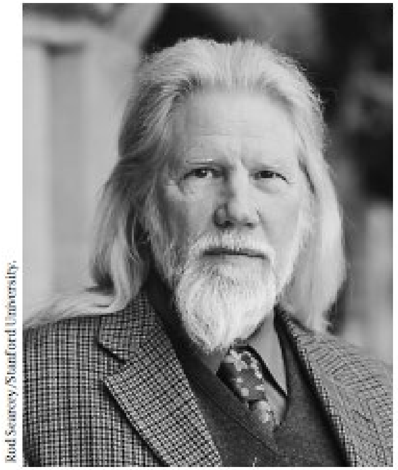
Session Key Establishment

Modular Arithmetic

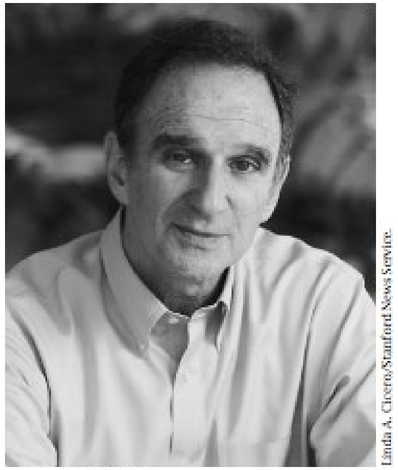
- Given g and prime p, compute: $g^1 \bmod p$, $g^2 \bmod p$, … $g^{100} \bmod p$
  - For p=11, g=10
    - $10^1 \bmod 11 = 10$, $10^2 \bmod 11 = 1$, $10^3 \bmod 11 = 10$, …
    - Produces cyclic group {10, 1} (order=2)
  - For p=11, g=7
    - $7^1 \bmod 11 = 7$, $7^2 \bmod 11 = 5$, $7^3 \bmod 11 = 2$, …
    - 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



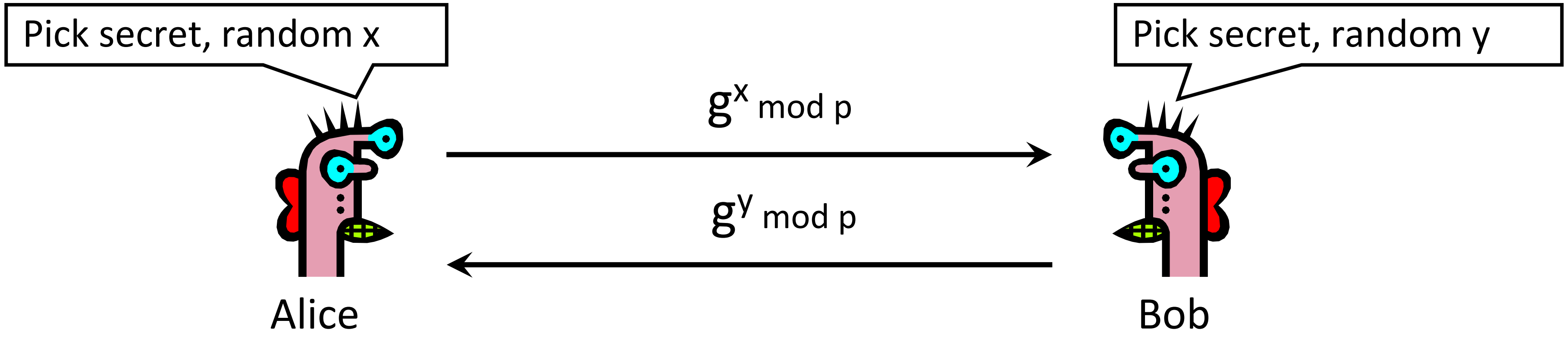
Whitfield Diffie



Martin E. Hellman

Diffie-Hellman Protocol (1976)

- Alice and Bob never met and share no secrets
- <u>Public</u> info: p and g
  - p is a large prime, g is a **generator** of Z_p*
    - Z_p*={1, 2 … p-1}; a is in Z_p* if there is an i such that a=g^i mod p



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

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

Example Diffie Hellman Computation

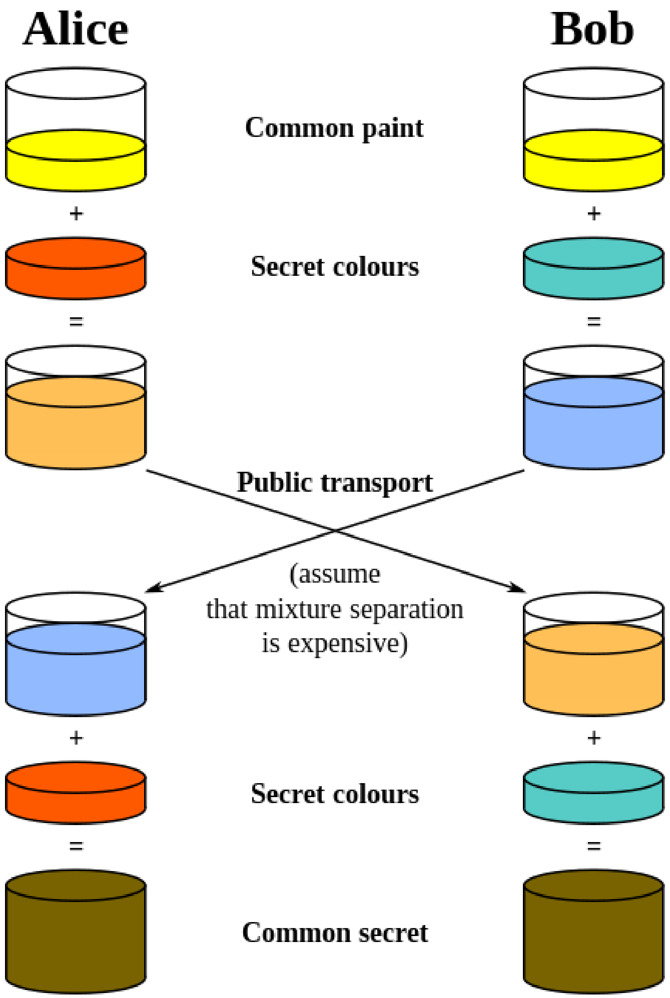
- PUBLIC
  - p = 11
  - g = 2
  - (g is a generator for group mod p)

- 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 mod p
g^y mod p

Common secret: g^{xy} mod p

[from Wikipedia]

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 Caveats (1)

- Assuming DDH problem is hard (depends on choice of parameters!), Diffie-Hellman protocol is a secure key establishment protocol against <u>passive</u> 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 <u>active</u> 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